Math. Prog. Comp. (2010) 2:167–201
DOI 10.1007/s12532-010-0016-2

FULL LENGTH PAPER

# Implementation of nonsymmetric interior-point methods for linear optimization over sparse matrix cones

**Martin S. Andersen · Joachim Dahl · Lieven Vandenberghe**

Received: 31 July 2009 / Accepted: 2 August 2010 / Published online: 26 August 2010


**Abstract** We describe an implementation of nonsymmetric interior-point methods for linear cone programs defined by two types of matrix cones: the cone of positive semidefinite matrices with a given chordal sparsity pattern and its dual cone, the cone of chordal sparse matrices that have a positive semidefinite completion. The implementation takes advantage of fast recursive algorithms for evaluating the function values and derivatives of the logarithmic barrier functions for these cones. We present experimental results of two implementations, one of which is based on an augmented system approach, and a comparison with publicly available interior-point solvers for semidefinite programming.

**Mathematics Subject Classification (2000)** 90-08 Mathematical Programming - computational methods · 90C06 Mathematical Programming - large-scale · 90C22 Mathematical Programming - semidefinite programing · 90C25 Mathematical Programming - convex programming · 90C51 Mathematical Programming - interior-point methods

The authors' (Andersen and Vandenberghe) research was supported in part by NSF grants ECS-0524663 and ECCS-0824003.

M. S. Andersen (✉) · L. Vandenberghe
Electrical Engineering Department, University of California, Los Angeles, USA
e-mail: msa@ee.ucla.edu

L. Vandenberghe
e-mail: vandenbe@ee.ucla.edu

J. Dahl
MOSEK ApS, Fruebjergvej 3, 2100 Copenhagen Ø, Denmark
e-mail: dahl.joachim@gmail.com

## 1 Introduction

Despite the progress made in the last 15 years, exploiting sparsity in semidefinite programming remains an important challenge. A fundamental difficulty is that the variable $X$ in the standard form semidefinite program (SDP)

$$\begin{array}{ll} \text{minimize} & C \bullet X \\ \text{subject to} & A_i \bullet X = b_i, \quad i = 1, \ldots, m, \\ & X \succeq 0 \end{array} \tag{1a}$$

is generally dense, even when the coefficients $A_i$, $C$ are sparse with a common sparsity pattern. This complicates the implementation of primal and primal–dual interior-point methods. Exploiting sparsity in dual methods is more straightforward, because the slack variable $S$ in the dual problem

$$\begin{array}{ll} \text{maximize} & b^T y \\ \text{subject to} & \displaystyle\sum_{i=1}^{m} y_i A_i + S = C \\ & S \succeq 0 \end{array} \tag{1b}$$

has the same (aggregate) sparsity pattern as $C$ and the matrices $A_i$. However computing the gradient and Hessian of the dual logarithmic barrier function requires the inverse of $S$, which is in general a dense matrix.

Fukuda and Nakata et al. [18,38], Burer [9], and Srijuntongsiri and Vavasis [48] propose to pose the problems as optimization problems in the subspace of symmetric matrices with a given sparsity pattern. Specifically, assume that $C, A_1, \ldots A_m \in \mathbf{S}^n_V$, where $\mathbf{S}^n_V$ denotes the symmetric matrices of order $n$ with sparsity pattern $V$. Then the pair of SDPs (1a)–(1b) is equivalent to

$$\begin{array}{lll} \text{P:} & \text{minimize} & C \bullet X \\ & \text{subject to} & A_i \bullet X = b_i, \quad i = 1, \ldots, m \\ & & X \succeq_{\mathrm{c}} 0 \end{array} \qquad \begin{array}{lll} \text{D:} & \text{maximize} & b^T y \\ & \text{subject to} & \displaystyle\sum_{i=1}^{m} y_i A_i + S = C \\ & & S \succeq 0, \end{array} \tag{2}$$

with primal variable $X \in \mathbf{S}^n_V$ and dual variables $y \in \mathbf{R}^m$, $S \in \mathbf{S}^n_V$. The inequality $X \succeq_{\mathrm{c}} 0$ means that the sparse matrix $X$ has a positive semidefinite completion ($X$ is the projection on $\mathbf{S}^n_V$ of a positive semidefinite matrix). The equivalence between (2) and (1a)–(1b) is easily established: if $X$ is optimal in (2), then any positive semidefinite completion of $X$ is optimal in (1a). Conversely, if $X$ is optimal in (1a), then $P_V(X)$, the projection of $X$ on $\mathbf{S}^n_V$, is optimal in (2). An important difference between (2) and (1a)–(1b) is that different types of inequalities are used in the primal and dual problems of (2).

By formulating the SDPs (1a)–(1b) as optimization problems in $\mathbf{S}^n_V$, we achieve a dimension reduction from $n(n+1)/2$ (the dimension of the space $\mathbf{S}^n$ of symmetric

matrices of order $n$) to $|V|$, the number of lower-triangular nonzeros in $V$. It is reasonable to expect that this can reduce the linear algebra complexity of interior-point methods for these problems. We will see that this is the case for methods based on primal or dual scaling, if the sparsity pattern $V$ is *chordal*. The reduction in complexity follows from efficient methods for evaluating the primal and dual barrier functions for the problems (2), their gradients, and Hessians [16]. The purpose of the paper is to describe an implementation of primal and dual path-following methods for problems (2), based on the chordal sparse matrix methods, and to present results of an extensive numerical comparison with the existing interior-point packages for semidefinite programming.[1]

**Outline** Section 2 provides some background on cone programming, additional motivation for the problems (2), and an overview of related work. In Sect. 3 we introduce the properties of chordal sparse matrices needed for the paper, and give a summary of the chordal matrix methods from [16]. In Sects. 4 and 5 we describe the interior-point method used in the numerical experiments. Section 6 contains the numerical results.

**Notation** $\mathbf{S}^n$ is the set of symmetric matrices of order $n$. $\mathbf{S}^n_+ = \{X \in \mathbf{S}^n \mid X \succeq 0\}$ and $\mathbf{S}^n_{++} = \{X \in \mathbf{S}^n \mid X \succ 0\}$ are the sets of positive semidefinite, respectively, positive definite matrices of order $n$. The notation $S \bullet X = \mathbf{tr}(SX) = \sum_{i,j=1}^n S_{ij}X_{ij}$ denotes the standard inner product of symmetric matrices of order $n$.

A sparsity pattern of a symmetric matrix is defined by the set $V$ of positions $(i, j)$ where the matrix is allowed to be nonzero, i.e., $X \in \mathbf{S}^n$ has sparsity pattern $V$ if $X_{ij} = X_{ji} = 0$ for $(i, j) \notin V$. It is assumed that all the diagonal entries are in $V$. Note that the entries of the matrix in $V$ are allowed to be zero as well. In that case we refer to them as *numerical zeros*, as opposed to the *structural zeros* in the sparsity pattern. The number of nonzero elements in the lower triangle of $V$ is denoted $|V|$.

$\mathbf{S}^n_V$ is the subspace of $\mathbf{S}^n$ of matrices with sparsity pattern $V$. $\mathbf{S}^n_{V,+}$ and $\mathbf{S}^n_{V,++}$ are the sets of positive semidefinite and positive definite matrices in $\mathbf{S}^n_V$. The projection $Y$ of a matrix $X \in \mathbf{S}^n$ on the subspace $\mathbf{S}^n_V$ is denoted $Y = P_V(X)$, i.e., $Y_{ij} = X_{ij}$ if $(i, j) \in V$ and otherwise $Y_{ij} = 0$.

$\mathbf{S}^n_{V,c+} = \{P_V(X) \mid X \succeq 0\}$ is the cone of matrices in $\mathbf{S}^n_V$ that have a positive semidefinite completion, and $\mathbf{S}^n_{V,c++}$ is the interior of $\mathbf{S}^n_{V,c+}$. The inequalities $\succeq_c$ and $\succ_c$ denote (strict) matrix inequality with respect to $\mathbf{S}^n_{V,c+}$, i.e., $X \succeq_c 0$ if and only if $X \in \mathbf{S}^n_{V,c+}$ and $X \succ_c 0$ if and only if $X \in \mathbf{S}^n_{V,c++}$. The functions $\phi$ and $\phi_c$ are logarithmic barrier functions for $\mathbf{S}^n_{V,+}$ and $\mathbf{S}^n_{V,c+}$, and are defined in Sect. 3.2 (Eqs. (6), (13)).

## 2 Cone programming

The optimization problems (2) are an example of a pair of primal and dual *cone programs*

$$\begin{array}{lll} \text{P:} & \text{minimize} & \langle c, x\rangle \\ & \text{subject to} & \mathcal{A}(x) = b \\ & & x \succeq_K 0. \end{array} \qquad \begin{array}{lll} \text{D:} & \text{maximize} & b^T y \\ & \text{subject to} & \mathcal{A}_{\text{adj}}(y) + s = c \\ & & s \succeq_{K^*} 0. \end{array} \tag{3}$$

[1] The software and benchmarks used in the paper are available at http://abel.ee.ucla.edu/smcp.

The variables $x$, $s$ in these problems are vectors in a vector space $E$, with inner product $\langle u, v\rangle$. The inequality $x \succeq_K 0$ means $x \in K$, where $K$ is a closed, convex, pointed cone with nonempty interior. The inequality $s \succeq_{K^*} 0$ means that $s$ is in the dual cone $K^* = \{s \mid \langle x, s\rangle \geq 0 \text{ for all } x \in K\}$. The mapping $\mathcal{A}$ in the primal equality constraint is a linear mapping from $E$ to $\mathbf{R}^m$, and $\mathcal{A}_{\text{adj}}$ is its adjoint, defined by the identity $u^T\mathcal{A}(v) = \langle \mathcal{A}_{\text{adj}}(u), v\rangle$.

When $K$ is the nonnegative orthant, the two problems reduce to a standard form linear program (LP) and its dual. Cone programming therefore provides a natural format for studying extensions of interior-point methods from linear programming to convex optimization [10], [39, Sect. 4], [42, Sect. 3]. For this reason, the cone program model of convex optimization is widely used in the recent literature on interior-point methods.

**Three symmetric cones** The research on algorithms for cone programming has focused almost exclusively on two nonpolyhedral cones: the second-order cone and the positive semidefinite cone [3,5,11,35,45,46,49,50,56]. These two cones, as well as the nonnegative orthant, are self-dual, i.e., $K = K^*$. The interest in the second-order and semidefinite cones is motivated by the possibility of formulating symmetric primal–dual algorithms that extend primal–dual algorithms for linear programming [55]. In linear programming these methods are known to be numerically more stable than purely primal or dual algorithms, and the same benefits are believed to hold for second-order cone and semidefinite programming as well. The special status of the nonnegative orthant, second-order cone, and positive semidefinite cone is further supported by the fact that they are not only self-dual, but *self-scaled* or *symmetric*, a property needed for the existence of the Nesterov-Todd symmetric primal–dual search direction [26,40,41].

The restriction of cone programming software to the three symmetric cones limits the class of convex problems that can be solved, but the impact is small in practice. With a few exceptions (for example, geometric programming) most nonlinear convex constraints encountered in practice can be expressed as second-order cone or semidefinite constraints [2,8,10,39,52]. In fact, YALMIP and CVX, two modeling packages for general convex optimization, are based on second-order cone and semidefinite programming solvers [21,22,32,33].

**Nonsymmetric sparse matrix cones** In the three-cone format used by SDP packages, any constraint that cannot be expressed as componentwise linear inequalities or second-order cone constraints must be converted to a semidefinite cone constraint. This has surprising consequences for the complexity of handling certain types of constraints.

Consider for example an SDP in which the coefficient matrices $A_i$, $C$ are banded with bandwidth $2w + 1$. If $w = 0$ (diagonal matrices), the problem reduces to an LP and the cost of solving it by an interior-point method increases linearly in $n$. (For dense problems, the cost per iteration is $O(m^2n)$ operations.) If $w > 0$, the band SDP cannot be cast as an LP or a second-order cone program (SOCP), and must be solved as an SDP. However, as we will see in Sect. 6.1, the cost per iteration of SDP solvers increases at least quadratically with $n$. This is surprising, because one would expect

the complexity to be close to the complexity of an LP, i.e., linear in $n$ for fixed $m$ and $w$.

A similar example illustrates the gap in complexity between second-order cone and semidefinite programming. It is well known that a Euclidean norm constraint $\|Ax + b\|_2 \leq t$ is equivalent to a linear matrix inequality (LMI) with arrow structure:

$$\begin{bmatrix} t & (Ax+b)^T \\ Ax+b & tI \end{bmatrix} \succeq 0.$$

This formulation is not recommended in practice, since the constraint is handled more efficiently as a second-order cone constraint [2]. However, one often encounters extensions with 'block arrow' structure. For example, a matrix norm constraint $\|A(x) + B\|_2 \leq t$, where $A(x) = x_1 A_1 + \cdots + x_m A_m$ and $A_i \in \mathbf{R}^{p\times q}$ is equivalent to a $(p+q) \times (p+q)$ LMI

$$\begin{bmatrix} tI & (A(x)+B)^T \\ A(x)+B & tI \end{bmatrix} \succeq 0.$$

Block-arrow structure is also common in robust optimization [17,24]. Since the block-arrow constraint cannot be reduced to a second-order cone constraint, it must be handled via its SDP formulation. This makes problems with matrix norm constraints substantially more difficult to solve than problems with Euclidean norm constraints, even when $q$ is small (see Sect. 6.2 for numerical experiments).

Band and block-arrow sparsity patterns are two examples of chordal structure. As the experiments in Sect. 6 will show, handling these constraints as nonsymmetric cone constraints results in a complexity per iteration that is linear in $n$, if the other dimensions are fixed. This provides one motivation for developing interior-point methods for the sparse matrix cone programs (2) with chordal sparsity patterns: the chordal sparse matrix cones form a family of useful convex cones that can be handled efficiently, without incurring the overhead of the embedding in the positive semidefinite cone. Since block-diagonal combinations of chordal sparsity patterns are also chordal, a solver that efficiently handles a single matrix inequality with a general chordal pattern applies to a wide variety of convex optimization problems. Moreover, general (non-chordal) sparsity patterns can often be embedded in a chordal pattern by adding a relatively small number of nonzeros (see Sect. 5.3).

**Related work** Chordality is a fundamental property in sparse matrix theory, and its role in sparse semidefinite programming has been investigated by several authors. The first papers to point out the importance of chordal sparsity in semidefinite programming were by Fukuda et al. [18] and Nakata et al. [38]. Two techniques are proposed in these papers. The first technique exploits chordal sparsity to reformulate an SDP with a large matrix variable as a problem with several smaller diagonal blocks and additional equality constraints. This is often easier to solve using standard semidefinite programming algorithms. The second method is a primal–dual path-following method for the optimization problems (2). The algorithm uses the HRVW/KSH/M search direction for general semidefinite programming [27,30,34], but applies it to

the maximum determinant positive definite completion of the primal variable $X$. The authors show that the search direction can be computed without explicitly forming the completion of $X$. This method has been implemented in the SDPA- C package.

Burer's algorithm [9] is a nonsymmetric primal–dual path-following method for the pair of cone programs (2). It is based on a formulation of the central path equations in terms of the Cholesky factors of the dual variable $S$ and the maximum determinant completion of the primal variable $X$. Linearizing the reformulated central path equations results in a new primal–dual search direction. The resulting algorithm is shown to have a polynomial-time worst-case complexity.

It is well known that positive definite matrices with chordal sparsity have a Cholesky factorization with zero fill-in. This provides a fast method for computing the standard logarithmic barrier function for the dual problem in (2), and via the chain rule, also for its gradient and Hessian. Srijuntongsiri and Vavasis [48] exploit this property in the computation of the dual Newton direction, by applying ideas from automatic differentiation in reverse mode. They also describe a fast algorithm for computing the primal barrier function, defined as the Legendre transform of the dual logarithmic barrier function, and its gradient and Hessian. The algorithm is derived from explicit formulas for the maximum determinant positive definite completion [25].

Most semidefinite programming solvers also incorporate techniques for exploiting sparsity that are not directly related to chordal sparsity. For example, Fujisawa et al. [19] present optimized methods for exploiting sparsity of the matrices $A_i$ when computing the quantities $H_{ij} = A_i \bullet (UA_jV), i, j = 1, \ldots, m$, with $U, V$ dense. The matrix $H$ is known as the Schur complement matrix, and its computation is a critical step in interior-point methods for semidefinite programming. An implementation of the techniques in [19] is available in the SDPA package. Other recent work has focused on exploiting sparsity in specific classes of SDPs (notably, SDPs derived from sum-of-squares relaxations of polynomial optimization problems [54]), and types of sparsity that ensure sparsity of the Schur complement matrix [29].

## 3 Chordal matrix cones

In this section we first review the definition and basic properties of chordal sparsity patterns [7,16,18,38]. We then summarize the chordal matrix algorithms of [16].

### 3.1 Chordal sparsity

A symmetric sparsity pattern $V$ of order $n$ can be represented by an undirected graph $\mathcal{G}_V$ with nodes $1, \ldots, n$, and edges between two nodes $i$ and $j$ $(i \neq j)$ if there is a nonzero in positions $i, j$ and $j, i$. (Although all the diagonal entries are assumed to be nonzero, the edges $(i, i)$ are not included in the graph $\mathcal{G}_V$.) Connected components of $\mathcal{G}_V$ correspond to blocks in a block diagonal sparsity pattern. The sparsity pattern is *chordal* if the graph $\mathcal{G}_V$ is chordal, i.e., every cycle of length greater than three has a chord (an edge joining nonconsecutive nodes of the cycle). Figure 1 shows three examples of chordal sparsity patterns.

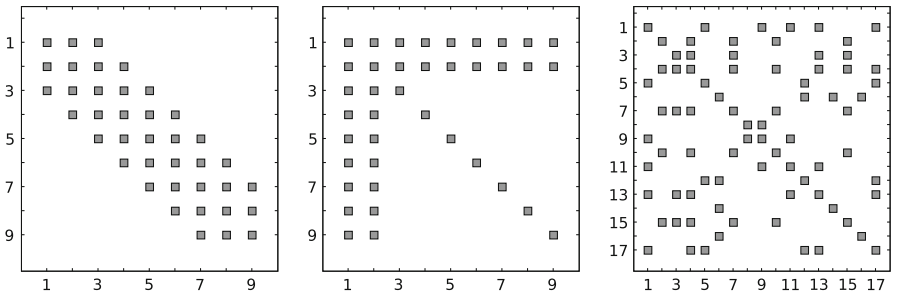

**Fig. 1** Examples of chordal sparsity patterns

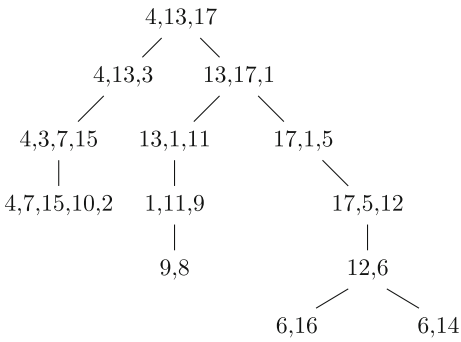


**Fig. 2** Clique tree of the third chordal sparsity pattern in Fig. 1

In the rest of this section we assume for simplicity that $\mathcal{G}_V$ is connected. However all algorithms extend in a straightforward manner to graphs consisting of several connected chordal components.

A *clique* in $\mathcal{G}_V$ is a set of nodes that defines a maximal complete subgraph. The nodes in a clique correspond to a dense principal subblock in the sparsity pattern. The cliques can be represented by a weighted undirected graph, with the cliques as its nodes, an edge between two cliques $W_i, W_j$ with a nonempty intersection, and a weight for edge $(W_i, W_j)$ equal to the cardinality of $W_i \cap W_j$. A *clique tree* is a maximum weight spanning tree of the clique graph. Figure 2 shows a clique tree for the third sparsity pattern in Fig. 1. Clique trees of chordal graphs can be efficiently computed by the *maximum cardinality search* algorithm [43,44,51]. This algorithm also provides an efficient (linear-time) test for chordality.

The useful properties of chordal sparsity patterns follow from a basic property known as the *running intersection property* [7]. Suppose $\mathcal{G}_V$ has $l$ cliques $W_1, \ldots, W_l$, numbered so that $W_1$ is the root of a clique tree, and every parent in the tree has a lower index than its children. We will refer to this as a *topological ordering* of the cliques. Define $U_1 = \emptyset$, $V_1 = W_1$, and, for $i = 2, \ldots, l$,

$$U_i = W_i \cap (W_1 \cup W_2 \cup \cdots \cup W_{i-1}), \quad V_i = W_i \backslash (W_1 \cup W_2 \cup \cdots \cup W_{i-1}). \tag{4}$$

The sets $V_i$ are sometimes called the *residuals*, and the sets $U_i$ the *separators* of the clique tree. Then the running intersection property states that

$$U_i = W_i \cap W_{\mathrm{par}(i)}, \quad V_i = W_i \setminus W_{\mathrm{par}(i)},$$

where $W_{\mathrm{par}(i)}$ is the parent of $W_i$ in the clique tree.

### 3.2 Chordal matrix algorithms

In this section we list a number of sparse matrix problems that can be solved by specialized methods if the underlying sparsity pattern is chordal. The algorithms consist of one or two recursions over the clique tree. Each recursion traverses the cliques in the tree in topological order (starting at the root) or reverse topological order. We omit the details of the algorithms, which can be found in [16]. A software implementation is available in the CHOMPACK library [15].

**Cholesky factorization** Positive definite matrices with chordal sparsity patterns have a Cholesky factorization with zero fill-in [7,43]: If $S \in \mathbf{S}^n_{V,++}$, then there exists a permutation matrix $P$ and a lower triangular matrix $L$ such that

$$P^T S P = L L^T, \tag{5}$$

and $L + L^T$ has the sparsity pattern $V$. This is perhaps the most important property of chordal graphs, and the basis of the algorithms described below. The permutation $P$ is called a *perfect elimination ordering*, and is easily derived from a clique tree. The factorization algorithm follows a recursion on the clique tree and visits the cliques in reverse topological order.

**Value and gradient of dual barrier** The Cholesky factorization provides an efficient method for evaluating the logarithmic barrier function for the cone $\mathbf{S}^n_{V,+}$, defined as

$$\phi : \mathbf{S}^n_V \to \mathbf{R}, \quad \phi(S) = -\log\det S, \quad \mathbf{dom}\,\phi = \mathbf{S}^n_{V,++}. \tag{6}$$

To evaluate $\phi(S)$, we compute the Cholesky factorization (5) and evaluate

$$\phi(S) = -2\sum_{i=1}^{n} \log L_{ii}.$$

The gradient of $\phi$ is given by

$$\nabla\phi(S) = -P_V(S^{-1}). \tag{7}$$

Although $S^{-1}$ is generally dense, its projection on the sparsity pattern can be computed from the Cholesky factorization of $S$ without computing any other entries of $S^{-1}$. The algorithm is recursive and visits the nodes of the clique tree in topological order.

**Hessian and inverse Hessian of dual barrier** The Hessian of $\phi$ at $S$, applied to a matrix $Y \in \mathbf{S}_V^n$, is given by

$$\nabla^2\phi(S)[Y] = P_V(S^{-1}YS^{-1}). \tag{8}$$

Evaluating this expression efficiently for large sparse matrices requires methods for computing the projection of $S^{-1}YS^{-1}$ on $\mathbf{S}_V^n$, without, however, computing $S^{-1}YS^{-1}$. This can be accomplished by exploiting chordal structure. The expression (8) can be evaluated from the Cholesky factorization of $S$ and the projected inverse $P_V(S^{-1})$, via two recursions on the clique tree, a first recursion that visits the cliques in reverse topological order, followed by a second recursion that visits the cliques in topological order.

The two recursions essentially form a pair of adjoint linear operators, and the algorithm for applying the Hessian can be interpreted as evaluating the Hessian in a factored form,

$$\nabla^2\phi(S)[Y] = \mathcal{L}_{\text{adj}}(\mathcal{L}(Y)), \tag{9}$$

by first evaluating a linear mapping $\mathcal{L}$ via an algorithm that visits the cliques in reverse topological order, followed by the evaluation of the adjoint $\mathcal{L}_{\text{adj}}$ via an algorithm that follows the topological order.

Furthermore, the factor $\mathcal{L}$ of the Hessian is easily inverted, and this provides a method for evaluating

$$\nabla^2\phi(S)^{-1}[Y] = \mathcal{L}^{-1}(\mathcal{L}_{\text{adj}}^{-1}(Y)) \tag{10}$$

(equivalently, for solving the linear equation $\nabla^2\phi(S)[U] = Y$) at the same cost as the evaluation of $\nabla^2\phi(S)[Y]$.

**Maximum determinant positive definite completion** Chordal sparsity plays a fundamental role in the maximum determinant matrix completion problem: given a matrix $X \in \mathbf{S}_V^n$, find the positive definite solution $Z$ of the optimization problem

$$\begin{array}{ll} \text{maximize} & \log\det Z \\ \text{subject to} & P_V(Z) = X. \end{array} \tag{11}$$

If $V$ is chordal, then the solution (if it exists) can be computed from $X$ via closed-form expressions [4], [18, Sect. 2], [25], [31, page 146], [38], [53, Sect. 3.2]. An equivalent algorithm for computing the Cholesky factor of $W = Z^{-1}$ is outlined in [16].

It follows from convex duality that $W$ has the sparsity pattern $V$, and satisfies the nonlinear equation

$$P_V(W^{-1}) = X. \tag{12}$$

The maximum determinant completion algorithm can therefore be interpreted as a method for solving the nonlinear equation (12) with variable $W \in \mathbf{S}_V^n$.

**Value and gradient of primal barrier** As a logarithmic barrier function for the matrix cone

$$\mathbf{S}^n_{V,\mathrm{c}+} = \{X \in \mathbf{S}^n_V \mid X \succeq_\mathrm{c} 0\} = (\mathbf{S}^n_{V,+})^*$$

we can use the Legendre transform of the barrier $\phi$ of $\mathbf{S}^n_{V,+}$ [39, p.48]. For $X \succ_\mathrm{c} 0$, the barrier function is defined

$$\phi_\mathrm{c}(X) = \sup_{S \succ 0} \left(-X \bullet S - \phi(S)\right). \tag{13}$$

(This is the Legendre transform of $\phi$ evaluated at $-X$.) If the sparsity pattern is chordal, the optimization problem in the definition can be solved analytically. The solution is the positive definite matrix $S \in \mathbf{S}^n_V$ that satisfies

$$P_V(S^{-1}) = X, \tag{14}$$

and as we have seen in the previous paragraph, $S^{-1}$ is the maximum determinant positive definite completion of $X$. This provides an efficient method for evaluating $\phi_\mathrm{c}$: We first compute the Cholesky factorization $\hat{S} = LL^T$ of the solution $\hat{S}$ of the maximization problem in (13), or equivalently, the nonlinear equation (14), and then compute

$$\phi_\mathrm{c}(X) = \log\det \hat{S} - n = 2\sum_{i=1}^{n} \log L_{ii} - n.$$

It follows from properties of Legendre transforms that

$$\nabla\phi_\mathrm{c}(X) = -\hat{S}, \tag{15}$$

and, from (14), this implies $X \bullet \nabla\phi_\mathrm{c}(X) = -n$.

**Hessian and inverse Hessian of primal barrier** The Hessian of the primal barrier function is given by

$$\nabla^2\phi_\mathrm{c}(X) = \nabla^2\phi(\hat{S})^{-1}, \tag{16}$$

where $\hat{S}$ is the maximizer in the definition of $\phi_\mathrm{c}(X)$. This result follows from standard properties of the Legendre transform. We can therefore evaluate $\nabla\phi_\mathrm{c}(X)[Y]$ using the methods for evaluating the inverse Hessian of the dual barrier (10), and we can compute $\nabla^2\phi_\mathrm{c}(X)^{-1}[Y]$ using the algorithm for evaluating the dual Hessian.

## 4 Primal and dual Newton systems

### 4.1 Central path

Interior-point methods follow the central path to guide the iteration towards the solution. For the primal–dual pair of cone programs (2) the central path is defined as the set of points $X \succ_c 0$, $y$, $S \succ 0$ that satisfy

$$A_i \bullet X = b_i, \quad i = 1, \ldots, m, \qquad \sum_{i=1}^{m} y_i A_i + S = C, \qquad S = -\mu \nabla \phi_c(X), \tag{17}$$

where $\mu$ is a positive parameter. An equivalent definition is

$$A_i \bullet X = b_i, \quad i = 1, \ldots, m, \qquad \sum_{i=1}^{m} y_i A_i + S = C, \qquad X = -\mu \nabla \phi(S). \tag{18}$$

The equivalence follows from properties of Legendre transform pairs. Interior-point methods that compute search directions based on linearizing (17) are called *primal scaling* methods, and methods based on linearizing (18) are called *dual scaling methods*. We will refer to the corresponding linearized equations as primal or dual Newton equations.

The solution of the Newton equations forms the bulk of the computation in any interior-point method. In this section we describe methods for solving the Newton equations when $V$ is chordal.

### 4.2 Primal scaling methods

Primal scaling directions are obtained by linearizing (17) around a triplet $X \succ_c 0$, $y$, $S \succ 0$ (the current primal and dual iterates in an interior-point method). Replacing $X$, $y$, $S$ with $X+\Delta X$, $y+\Delta y$, $S+\Delta S$ in (17), linearizing the third equation, and eliminating $\Delta S$ gives

$$A_i \bullet \Delta X = r_i, \quad i = 1, \ldots, m, \qquad \sum_{i=1}^{m} \Delta y_i A_i - \mu \nabla^2 \phi_c(X)[\Delta X] = R, \tag{19}$$

where $r_i = b_i - A_i \bullet X$ and

$$R = C - \sum_{i=1}^{m} y_i A_i + \mu \nabla \phi_c(X).$$

The variables are $\Delta X \in \mathbf{S}_V^n$, $\Delta y \in \mathbf{R}^m$, so this is a set of $|V|+m$ equations in $|V|+m$ variables (where $|V|$ is the number of lower triangular nonzeros in $V$). By eliminating

$\Delta X$ we can further reduce the Newton equation to

$$H\Delta y = g, \tag{20}$$

where $H$ is the positive definite symmetric matrix with entries

$$H_{ij} = A_i \bullet \left(\nabla^2\phi_c(X)^{-1}[A_j]\right), \quad i, j = 1, \ldots, m, \tag{21}$$

and $g_i = \mu r_i + A_i \bullet (\nabla^2\phi_c(X)^{-1}[R])$.

We now outline two methods for solving (20) and will compare their practical performance in Sect. 6. Recall from Sect. 3 that $\nabla^2\phi_c(X) = \nabla^2\phi(\hat{S})^{-1}$ where $\hat{S}$ solves $P_V(\hat{S}^{-1}) = X$, and that the Cholesky factor $L$ of $\hat{S}$ is readily computed from $X$ using a recursive algorithm. The entries of $H$ can therefore be written

$$H_{ij} = A_i \bullet (\nabla^2\phi(\hat{S})[A_j]), \quad i, j = 1, \ldots, m. \tag{22}$$

*Method 1: Cholesky factorization*

The first method explicitly forms $H$, column by column, and then solves (20) using a dense Cholesky factorization. We distinguish two techniques for computing column $j$. The first technique is a straightforward evaluation of (22). It first computes $\nabla^2\phi(\hat{S})[A_j]$ and then completes the lower-triangular part of column $j$ of $H$ by making inner products with the matrices $A_i$:

$T1:$ $U := \nabla^2\phi_c(X)^{-1}[A_j]$
**for** $i = j$ to $m$ **do**
$H_{ij} := A_i \bullet U$
**end for**

Additional sparsity of $A_i$, relative to $V$, can be exploited in the inner products, but additional sparsity in $A_j$ is not easily exploited. The matrix $U$ is a dense matrix in $\mathbf{S}^n_V$, and $A_j$ is handled as a dense element in $\mathbf{S}^n_V$.

The second technique is useful when $A_j$ is very sparse, relative to $V$, and has only a few nonzero columns. We write $A_j$ as

$$A_j = \sum_{(p,q)\in I_j} (A_j)_{pq} e_p e_q^T,$$

where $e_k$ is the $k$th unit vector in $\mathbf{R}^n$ and $I_j$ indexes the nonzero entries in $A_j$. The expression for $H_{ij}$ in (20) can then be written as

$$
\begin{aligned}
H_{ij} &= \sum_{(p,q)\in I_j} (A_j)_{pq} \left( A_i \bullet (\nabla^2 \phi_c(X)^{-1}[e_p e_q^T]) \right) \\
&= \sum_{(p,q)\in I_j} (A_j)_{pq} \left( A_i \bullet P_V(\hat{S}^{-1} e_p e_q^T \hat{S}^{-1}) \right) \\
&= \sum_{(p,q)\in I_j} (A_j)_{pq} \left( A_i \bullet (L^{-T} L^{-1} e_p e_q^T L^{-T} L^{-1}) \right).
\end{aligned}
$$

If $A_j$ has only a few nonzero columns, it is advantageous to precompute the vectors $u_k = L^{-T} L^{-1} e_k$ that occur in this expression. This results in the following algorithm for computing column $j$ of $H$.

$$
\begin{aligned}
T2: \quad & \text{Solve } LL^T u_k = e_k \text{ for } k \in \{i \mid A_j e_i \neq 0\} \\
& \textbf{for } i = j \text{ to } m \textbf{ do} \\
& \quad H_{ij} := \sum_{(p,q)\in I_j} (A_j)_{pq} u_q^T A_i u_p \\
& \textbf{end for}
\end{aligned}
$$

Since $A_j$ is symmetric, the summation can easily be modified to use only the lower triangular entries of $A_j$. If $A_j$ has $\zeta_j$ nonzero columns, T2 requires that $\zeta_j$ vectors $u_k$ be computed and stored in order to form column $j$ of $H$. Since this is generally the most expensive part of T2, the technique is efficient only when $\zeta_j$ is small.

T2 is somewhat similar to a technique used in SDPA-C [38, p. 316]. However SDPA-C only stores two vectors at the time, and forming the $j$th column of $H$ requires that $2\zeta_j$ systems of the form $LL^T x = b$ be solved (twice the number of T2). It is also worth mentioning that low-rank techniques such as those used in DSDP and SDPT3 provide a more general and often faster alternative to T2. However, these low rank techniques require that a low-rank factorization of the data matrices be computed and stored as a preprocessing step. Furthermore, we remark that in its current form T2 does not exploit any block structure of $\hat{S}$.

A threshold on the number of nonzero columns $\zeta_j$ can be used as a heuristic for choosing between T1 and T2. Based on some preliminary experiments, we set the threshold to $n/10$ in the implementation used in Sect. 6 (in other words, we use T1 if $\zeta_j > n/10$ and T2 otherwise). Further experimentation would be needed to find better guidelines to automatically tune the threshold. Finally we note that the total flop count can be reduced by permuting the order of the data matrices.

*Method 2: QR factorization*

The second method for solving the reduced Newton equation (20) avoids the explicit calculation of $H$, by applying the factorization of the Hessian matrix in Sect. 3. Using the factorization $\nabla^2 \phi_c(X)^{-1} = \nabla^2 \phi(\hat{S}) = \mathcal{L}_{\text{adj}} \circ \mathcal{L}$, we can write

$$
H_{ij} = A_i \bullet \left( \nabla^2 \phi_c(X)^{-1}[A_j] \right) = \mathcal{L}(A_i) \bullet \mathcal{L}(A_j).
$$

The factorization allows us to express $H$ as $H = \tilde{A}^T \tilde{A}$ where $\tilde{A}$ is a $|V| \times m$ matrix with columns $\mathbf{vec}(\mathcal{L}(A_i))$. (The $\mathbf{vec}(\cdot)$ operator converts matrices in $\mathbf{S}_V^n$ to vectors containing the lower-triangular nonzeros, scaled so that $\mathbf{vec}(U_1)^T\,\mathbf{vec}(U_2) = U_1 \bullet U_2$ for all $U_1, U_2 \in \mathbf{S}_V^n$.) Instead of computing a Cholesky factorization of $H$ (constructed via (21) or as $\tilde{A}^T \tilde{A}$), we can compute a QR-decomposition of $\tilde{A}$ and use it to solve (20). This is important, because the explicit computation of $H$ is a source of numerical instability.

The second method is a variation of the *augmented systems* approach in linear programming interior-point methods. In the augmented systems approach, one computes the solution of the Newton equation

$$\begin{bmatrix} -D & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 \end{bmatrix} \tag{23}$$

(the linear programming equivalent of (19)) via an LDL$^{\mathrm{T}}$ or QR factorization, instead of eliminating $\Delta x$ and solving the 'Schur complement' equation $AD^{-1}A^T\Delta y = r_2 + AD^{-1}r_1$ by a Cholesky factorization [20], [55, p.220]. The augmented system (23) can be solved using a sparse LDL$^{\mathrm{T}}$ factorization (for sparse $A$), or via a QR decomposition of $\tilde{A} = D^{-1/2}A^T$ (for dense $A$). The augmented system method is known to be more stable than methods based on Cholesky factorization of $AD^{-1}A^T$. It is rarely used in practice, since it is slower for large sparse LPs than the Cholesky factorization of $AD^{-1}A^T$. In semidefinite programming, the loss of stability in forming $H$ is more severe than in linear programming [47]. This would make the augmented system approach even more attractive than for linear programming, but unfortunately the large size of the Newton equations make it very expensive. In our present context, however, the augmented system approach is often feasible, since we work in the subspace $\mathbf{S}_V^n$ and the row dimension of $\tilde{A}$ is proportional to $|V|$.

### 4.3 Dual scaling methods

The dual Newton equations can be derived in a similar way, by linearizing (18):

$$-\mu\nabla^2\phi(S)^{-1}[\Delta X] + \sum_{i=1}^{m} \Delta y_i A_i = R, \qquad A_i \bullet \Delta X = r_i, \quad i = 1, \ldots, m \tag{24}$$

with $r_i = b_i - A_i \bullet X$ and

$$R = C - \sum_{i=1}^{m} y_i A_i - 2S + \mu\nabla^2\phi(S)^{-1}[X].$$

Eliminating $\Delta X$ and $\Delta S$ gives a set of linear equations (21) with

$$H_{ij} = A_i \bullet (\nabla^2\phi(S)[A_j]), \quad i, j = 1, \ldots, m.$$

This matrix has the same structure as (22). Therefore, the same methods can be used to compute the dual scaling direction as for the primal scaling direction and the complexity is exactly the same.

### 4.4 Complexity

The cost of solving the Newton system (19) is dominated by the cost of solving (20). Method 1 solves (20) by explicitly forming $H$ and then applying a Cholesky factorization. Method 2 avoids explicitly forming $H$, by applying a QR factorization to a matrix $\tilde{A}$ that satisfies $H = \tilde{A}^T \tilde{A}$.

**Method 1** The cost of solving (20) via the Cholesky factorization of $H$ depends on the sparsity of the data matrices and on the techniques used to form $H$. The matrix $H$ is generally dense, and hence the cost of factorizing $H$ is $O(m^3)$. If all data matrices share the same sparsity pattern $V$ and are dense relative to $\mathbf{S}^n_V$, the cost of forming $H$ using technique T1 is $mK + O(m^2|V|)$ where $K$ is the cost of evaluating $\nabla^2\phi_c(X)^{-1}[A_j]$. If the data matrices $A_i$ are very sparse relative to $V$, the cost of computing $H$ using T1 is roughly $mK$.

With technique T2, the dominating cost of computing the columns of $H$ is solving the systems $LL^T u_k = e_k$ for $k \in \{i \mid A_j e_i \neq 0\}$. Thus, the cost mainly depends on $|V|$ and the number of nonzero columns in the data matrices. When the data matrices only have a small number of nonzeros, T2 is generally many times faster than T1.

The cost $K$ depends on the clique distribution (i.e., the structure of $V$) in a complicated way, but for special cases such as banded and arrow matrices we have $|V| = O(n)$ and $K = O(n)$, and in these special cases the cost of one iteration is *linear* in $n$.

**Method 2** Solving (20) via a QR decomposition of $\tilde{A}$ costs $O(mK)$ to form $\tilde{A}$ and $O(m^2|V|)$ to compute the QR decomposition. In particular, the cost of one iteration is also *linear* in $n$ for banded and arrow matrices, when the other dimensions are fixed. If the coefficients $A_i$ are dense relative to $V$, the cost of Method 2 is at most twice that of Method 1 when only T1 is used.

## 5 Implementation

The techniques described in the previous section can be used in any interior-point methods based on primal or dual scaling directions, for example, barrier methods, infeasible primal or dual path-following methods, or nonsymmetric primal–dual methods [36,37]. In this section we describe the algorithm used in the numerical experiments of Sect. 6. The algorithm is a feasible-start path-following algorithm.

### 5.1 Feasible-start primal scaling method

The algorithm is a variant of the barrier method. It is based on the interpretation of the central path as the set of solutions of the equality-constrained problem

$$\begin{aligned} &\text{minimize } C \bullet X + \mu\phi_{\mathrm{c}}(X) \\ &\text{subject to } A_i \bullet X = b_i, \quad i = 1, \ldots, m, \end{aligned}$$

as a function of $\mu > 0$. (The dual scaling variant of the algorithm is similar and will not be discussed here.) In each iteration several equations of the form

$$A_i \bullet \Delta X = 0, \quad i = 1, \ldots, m, \tag{25}$$

$$\sum_{i=1}^{m} \Delta y_i A_i + \Delta S = 0, \tag{26}$$

$$\mu\nabla^2\phi_{\mathrm{c}}(X)[\Delta X] + \Delta S = -R \tag{27}$$

are solved.

The data matrices $A_i$ are assumed to be linearly independent. We also assume that a chordal embedding is available and that a clique tree has been computed.

**Algorithm outline** The algorithm depends on parameters $\delta \in (0, 1)$, $\gamma \in (0, 1/2)$, $\beta \in (0, 1)$, and tolerances $\epsilon_{\mathrm{abs}}$ and $\epsilon_{\mathrm{rel}}$. (The values used in our experiments are $\delta = 0.9$, $\gamma = 0.1$, $\beta = 0.7$, $\epsilon_{\mathrm{abs}} = \epsilon_{\mathrm{rel}} = 10^{-7}$.) The algorithm also requires a strictly feasible starting point $X$ and an initial value of the positive parameter $\mu$ (see Sect. 5.2).

Repeat the following steps.

1. *Primal centering.* Solve (25) with $R = C + \mu\nabla\phi_{\mathrm{c}}(X)$ and denote the solution by $\Delta X_{\mathrm{cnt}}$, $\Delta y_{\mathrm{cnt}}$, $\Delta S_{\mathrm{cnt}}$. Evaluate the Newton decrement

$$\lambda = \left(\Delta X_{\mathrm{cnt}} \bullet \nabla^2\phi_{\mathrm{c}}(X)[\Delta X_{\mathrm{cnt}}]\right)^{1/2}.$$

If $\lambda \leq \delta$, set

$$S := C + \Delta S_{\mathrm{cnt}}, \quad y := \Delta y_{\mathrm{cnt}}$$

and proceed to step 2. Otherwise, conduct a backtracking line search to find a step size $\alpha$ that satisfies $X + \alpha\Delta X_{\mathrm{cnt}} \succ_{\mathrm{c}} 0$ and the Armijo condition

$$\frac{1}{\mu}(C \bullet (X + \alpha\Delta X_{\mathrm{cnt}})) + \phi_{\mathrm{c}}(X + \alpha\Delta X_{\mathrm{cnt}}) \leq \frac{1}{\mu}(C \bullet X) + \phi_{\mathrm{c}}(X) - \alpha\gamma\lambda^2. \tag{28}$$

Update $X$ as $X := X + \alpha\Delta X_{\mathrm{cnt}}$. Repeat step 1.

2. *Prediction step.* Solve (25) with $R = S$ and denote the solution by $\Delta X_{\mathrm{at}}$, $\Delta y_{\mathrm{at}}$, $\Delta S_{\mathrm{at}}$. Calculate

$$\hat{\mu} := \frac{(\tilde{X} + \alpha\Delta X_{\mathrm{at}}) \bullet (S + \alpha\Delta S_{\mathrm{at}})}{n} = (1 - \alpha)\frac{\tilde{X} \bullet S}{n}$$

where $\tilde{X} := X - \Delta X_{\text{cnt}}$ and

$$\alpha = 0.98 \cdot \sup \left\{ \alpha \in [0, 1) \;\middle|\; \tilde{X} + \alpha \Delta X_{\text{at}} \succ_{\text{c}} 0,\, S + \alpha \Delta S_{\text{at}} \succ 0 \right\}.$$

Solve (25) with $\mu = \hat{\mu}$ and $R = S + \hat{\mu} \nabla \phi_{\text{c}}(X)$. Conduct a backtracking line search to find a primal step size $\alpha_{\text{p}}$ such that $X + \alpha_{\text{p}} \Delta X \succ_{\text{c}} 0$ and the Armijo condition

$$\frac{1}{\hat{\mu}}(C \bullet (X + \alpha \Delta X)) + \phi_{\text{c}}(X + \alpha \Delta X) \leq \frac{1}{\hat{\mu}}(C \bullet X) + \phi_{\text{c}}(X) - \alpha \gamma \lambda^2$$

is satisfied, where $\lambda = (\Delta X \bullet \nabla^2 \phi_{\text{c}}(X)[\Delta X])^{1/2}$. Conduct a backtracking line search to find the dual step size

$$\alpha_{\text{d}} = \max_{k=0,1,\ldots} \{\beta^k \mid S + \beta^k \Delta S \succ 0\}, \tag{29}$$

where $\beta \in (0, 1)$ is an algorithm parameter. Update the variables

$$X := X + \alpha_{\text{p}} \Delta X, \quad y := y + \alpha_{\text{d}} \Delta y, \quad S := S + \alpha_{\text{d}} \Delta S. \tag{30}$$

3. *Stopping criteria.* Terminate if the (approximate) optimality conditions

$$X \bullet S \leq \epsilon_{\text{abs}} \quad \text{or} \quad \left( \min\{C \bullet X, -b^T y\} < 0,\; \frac{X \bullet S}{-\min\{C \bullet X, -b^T y\}} \leq \epsilon_{\text{rel}} \right).$$

are satisfied. Otherwise, set $\mu := (X \bullet S)/n$ and go to step 1.

*Remarks* The three systems of equations that are solved in each iteration have the same coefficient matrix (if we absorb the scalar $\mu$ in $\Delta X$) and therefore require only one factorization. They can be solved using either method 1 or method 2 from Sect. 4. We will refer to these two variants of the algorithm as M1 and M2, respectively. Moreover the right-hand side of the third system is a linear combination of the right-hand sides of the first two systems, so the solution can be obtained by combining the first two solutions.

The prediction direction in step 3 is based on the approximate tangent direction proposed by Nesterov [36,37], who refers to the intermediate point $\tilde{X} = X - \Delta X$ as a *lifting*. Here the approximate tangent is used only to calculate $\hat{\mu}$; the actual step made in step 2 is a centering step with centering parameter $\hat{\mu}$. The simplification in the definition of $\hat{\mu}$ follows from the identities

$$\Delta X_{\text{at}} \bullet \Delta S_{\text{at}} = 0, \quad \tilde{X} \bullet S + \tilde{X} \bullet \Delta S_{\text{at}} + S \bullet \Delta X_{\text{at}} = 0.$$

The algorithm can be improved in several ways, most importantly by allowing infeasible starting points and by making more efficient steps in the approximate tangent direction. However these improvements would be of no consequence for the two

questions we aim to answer in the experiments: how does the cost per iteration compare with general-purpose sparse semidefinite programming solvers and, secondly, how does the choice for primal scaling affect the accuracy that can be attained?

### 5.2 Phase I

The algorithm outlined in Sect. 5.1 is a feasible-start method and hence we need a feasible starting point. To this end we first solve the least-norm problem

$$\begin{array}{ll} \text{minimize} & \|X\|_F^2 \\ \text{subject to} & A_i \bullet X = b_i, \quad i = 1, \ldots, m. \end{array}$$

If the solution $X_{\text{ln}}$ satisfies $X_{\text{ln}} \succ_{\text{c}} 0$, we use it as the starting point. On the other hand, if $X_{\text{ln}} \not\succ_{\text{c}} 0$, we solve the phase I problem

$$\begin{array}{ll} \text{minimize} & s \\ \text{subject to} & A_i \bullet X = b_i, \quad i = 1, \ldots, m \\ & \mathbf{tr}(X) \leq M \\ & X + (s - \epsilon) I \succeq_{\text{c}} 0, \quad s \geq 0 \end{array} \tag{31}$$

where $\epsilon > 0$ is a small constant. The constraint $\mathbf{tr}(X) \leq M$ (with $M$ large and positive) is added to bound the feasible set. The optimal solution $X^\star$ is strictly feasible in (2) if $s^\star < \epsilon$. The phase I problem (31) can be cast in the standard form and solved using the feasible start algorithm.

### 5.3 Chordal embedding of non-chordal sparsity patterns

When the data matrices $A_i$, $C$ have a common chordal sparsity pattern $V$, we have seen that it is possible to formulate and take advantage of efficient algorithms for evaluating barriers and their first and second derivatives. For nonchordal sparsity patterns it is possible to exploit chordality by constructing a *chordal embedding* or *triangulation* of $V$, i.e., by adding edges to the graph to make it chordal. Chordal embeddings are not only useful for nonchordal sparsity patterns. By extending a chordal sparsity pattern it is sometimes possible to "shape" the clique tree to improve the computational efficiency of the algorithms. For example, merging cliques with large relative overlap often improves performance. A similar observation was made by Fukuda et al. [18] who demonstrated that with their conversion method, a balance between the number of cliques and the size of the cliques and the separator sets is critical to obtain good performance.

In practice a chordal embedding is easily constructed by computing a symbolic Cholesky factorization of the sparsity aggregate pattern. The amount of fill-in (i.e., the number of added edges) generally depends heavily on the ordering of the nodes, and different chordal embeddings can be obtained simply by reordering the nodes. In sparse matrix computations it is generally desirable to minimize fill-in since fill-in requires additional storage and often increases the time needed to solve a system of

**Table 1** DIMACS error measures for `control6` from SDPLIB

| Solver | $\epsilon_1$ | $\epsilon_3$ | $\epsilon_5$ | $\epsilon_6$ |
|---|---|---|---|---|
| M1 | 1.63e−07 | 0.00e+00 | 1.04e−05 | 6.79e−06 |
| M2 | 9.97e−14 | 0.00e+00 | 4.30e−10 | 3.63e−10 |
| CSDP | 5.67e−08 | 9.41e−09 | 3.66e−08 | 1.42e−08 |
| SDPA | 4.17e−07 | 1.81e−09 | 1.15e−06 | 1.03e−06 |
| SDPT3 | 3.50e−07 | 1.80e−09 | 7.80e−07 | 7.40e−07 |
| SEDUMI | 1.45e−06 | 0.00e+00 | −2.92e−08 | 3.28e−06 |

equations. However, when practical considerations such as data locality and cache efficiency are considered, minimizing fill-in is not necessarily optimal in terms of run-time complexity. This can be seen in BLAS-based supernodal Cholesky codes where additional fill-in may be created in order to increase the size of the supernodes (so-called relaxed supernodal amalgamation). Some common fill-in reducing ordering heuristics are the approximate minimum degree (AMD) ordering [1] and nested dissection (ND) [23].

### 5.4 Numerical stability

As mentioned in Sect. 4, the augmented systems approach used in M2 is more stable than methods based on the Cholesky factorization of the Schur complement matrix $H$ (method M1). To illustrate the difference we consider the problem `control6` from SDPLIB. This problem has a dual degenerate solution. Using default accuracy settings, the symmetric primal–dual codes CSDP, SDPA, SDPT3, and SEDUMI all stop prematurely because the Cholesky factorization of the Schur complement matrix fails near the solution where the Schur complement system is severely ill-conditioned. This is also the case for M1. However the augmented systems approach in M2 solves the problem to a high accuracy. Table 1 shows four of the DIMACS error measures [28] defined as

$$\epsilon_1 = \frac{\|r\|_2}{1 + \|b\|_\infty}, \quad \epsilon_3 = \frac{\|R\|_F}{1 + \|C\|_{\max}}, \quad \epsilon_5 = \frac{C \bullet X - b^T y}{1 + |C \bullet X| + |b^T y|},$$
$$\epsilon_6 = \frac{S \bullet X}{1 + |C \bullet X| + |b^T y|}$$

where $\|C\|_{\max} = \max_{i,j} |C_{ij}|$. (The remaining two DIMACS error measures are equal to zero for all the solvers.) Although the sparsity pattern associated with `control6` is not very sparse, the results in Table 1 demonstrate the benefit of the augmented systems approach in terms of numerical stability. Similar results can be observed for several other SDPLIB problems.

## 6 Numerical experiments

To assess the speed and accuracy of the algorithm described in Sect. 5.1, we have conducted a series of experiments using a preliminary implementation of the algorithm.

The implementation is in Python and C, and relies on two other Python packages, CVXOPT 1.1.2 [14] and CHOMPACK 1.1 [15]. We also tested a method based on dual scaling. These results are not included but were similar to the results for the primal scaling method.

The experiments consist of three families of randomly generated problems, selected sparse problems from SDPLIB [6], and a set of very large sparse SDPs. All the experiments were conducted on a desktop computer with an Intel Core 2 Quad Q6600 CPU (2.4 GHz), 4 GB RAM, and Ubuntu 9.10 (64-bit). Each problem instance was solved with the interior-point SDP solvers DSDP 5.8, SDPA- C 6.2.1, SDPT3 4.0b, and SEDUMI 1.2. The Matlab-based solvers SDPT3 and SEDUMI were run in Matlab R2008b while the other solvers are stand-alone and linked to ATLAS/LAPACK. It should be noted that DSDP reports "wall time" (i.e., real time) whereas the other solvers report CPU time. However, since a single threaded version of DSDP is used, the difference between wall time and CPU time is negligible in practice.

All the solvers were run with their default parameters and starting points. In our solver we used the tolerances $\epsilon_{\text{abs}} = \epsilon_{\text{rel}} = 10^{-7}$ in the exit conditions. One iterative refinement step is applied when solving the Newton systems using M2, and three iterative refinement steps are applied when using M1. We use the centering parameter $\delta = 0.9$ and the backtracking parameters $\beta = 0.7$ and $\gamma = 0.1$ in the line searches. The initial $\mu$ is 100.

In the following we use the AMD ordering to compute chordal embeddings when the sparsity pattern is nonchordal. Recall that M1 uses a combination of two techniques (referred to as T1 and T2 in Sect. 4.2) to form the Schur complement matrix. For each column of the Schur complement matrix a threshold is used to select between the two techniques. The threshold is based on the number of nonzero columns in $A_i$: T1 is used if $A_i$ has more than $0.1 \cdot n$ nonzero columns, and otherwise T2 is used. This criterion is a simple heuristic based on experimentation, and leaves room for improvement. The second method, M2, does not explicitly form the Schur complement matrix, and treats each of the data matrices $A_i$ as a dense element in $\mathbf{S}^n_V$.

The main purpose of the experiments is to compare the linear algebra complexity of the techniques described in the previous sections with existing interior-point solvers. For most test problems the number of iterations needed by the different solvers was comparable and ranged between 20 and 50 iterations.[2] We report the average time per iteration as well as the total time. The cost per iteration is arguably the fairest basis for a comparison, since the solvers use different stopping criteria, start at different initial points, and use different strategies for step size selection and for updating the central path parameter.

The solution times for M1 and M2 will not include the time spent in phase I. Most test problems did not require a phase I because the solution of the least-norm problem (31) happened to be strictly feasible. The few problems that did require a full phase I will be pointed out in the text.

[2] The results are available in full detail at http://abel.ee.ucla.edu/smcp/.

**Table 2** Time per iteration and total time for randomly generated band SDPs with $m = 100$ and $w = 5$

| $n$ | Time per iteration (s) | | | | | | Total time (s) | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | M1 | M2 | DSDP | SDPA-C | SDPT3 | SEDUMI | M1 | M2 | DSDP | SDPA-C | SDPT3 | SEDUMI |
| 100 | 0.12 | 0.05 | 0.20 | 5.1 | 0.20 | 0.09 | 4.2 | 1.8 | 6.3 | 117 | 3.0 | 1.2 |
| 200 | 0.23 | 0.10 | 1.2 | 20 | 0.46 | 0.36 | 9.3 | 4.3 | 41 | 510 | 7.9 | 5.7 |
| 400 | 0.48 | 0.22 | 9.0 | 86 | 1.5 | 2.2 | 18 | 8.3 | 314 | 2,065 | 23 | 35 |
| 800 | 1.0 | 0.45 | 132 | 576 | 5.4 | 16 | 41 | 18 | 4,766 | 13,247 | 91 | 249 |
| 1,600 | 2.0 | 0.98 | 3,988 | 4,312 | 28 | 171 | 94 | 46 | 171,500 | 116,426 | 500 | 3,086 |

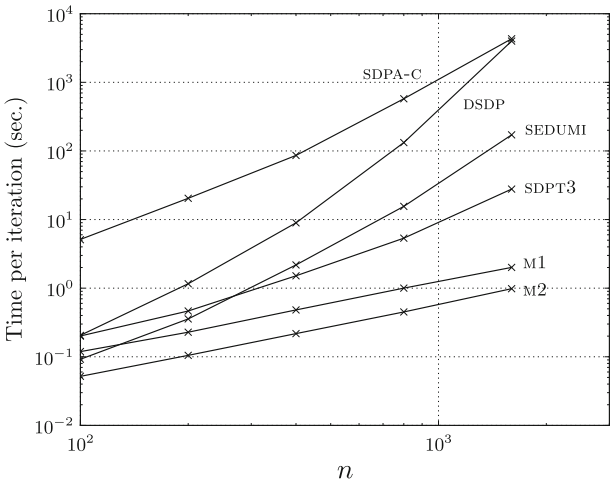


**Fig. 3** Time per iteration as a function of the problem parameter $n$. The cost of one iteration increases linearly with $n$ for M1 and M2 whereas the cost grows at least quadratically for the other solvers

### 6.1 SDPs with band structure

In the first experiment we consider randomly generated SDPs with band structure, i.e., the matrices $C, A_i \in \mathbf{S}^n$ $(i = 1, \ldots, m)$ are all banded with a common bandwidth $2w + 1$. The special case where all $A_i$ are diagonal $(w = 0)$ corresponds to a linear programming problem.

Table 2 shows the total time and the time per iteration as a function of $n$, with a fixed number of constraints $(m = 100)$ and fixed bandwidth $(w = 5)$. The time per iteration is also shown in Fig. 3. We see that, in terms of time per iteration, M1 and M2 scale linearly with $n$. The advantage is clear for large values of $n$. The high iteration times for DSDP and SDPA-C can be explained by implementation choices rather than fundamental limitations. The problem data in this experiment have full rank and as a consequence the low-rank techniques used in DSDP become expensive. Furthermore, recall that SDPA-C uses a technique similar to technique T2 described in Sect. 4.2. This is clearly inefficient when all the columns of the data matrices are nonzero.

**Table 3** Time per iteration and total time for randomly generated band SDPs with $n = 500$ and $w = 3$

| $m$ | Time per iteration (s) | | | | | | Total time (s) | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | M1 | M2 | DSDP | SDPA-C | SDPT3 | SEDUMI | M1 | M2 | DSDP | SDPA-C | SDPT3 | SEDUMI |
| 10 | 0.07 | 0.03 | 13 | 1.9 | 0.46 | 3.4 | 2.8 | 1.2 | 450 | 47 | 7.4 | 62 |
| 50 | 0.19 | 0.08 | 27 | 36 | 0.96 | 3.7 | 7.5 | 3.2 | 1,052 | 891 | 15 | 59 |
| 100 | 0.41 | 0.18 | 31 | 135 | 1.7 | 3.9 | 15 | 6.8 | 1,102 | 3,241 | 27 | 55 |
| 200 | 1.1 | 0.37 | 46 | 555 | 3.1 | 4.3 | 46 | 15 | 1,888 | 13,318 | 53 | 69 |
| 400 | 3.5 | 0.80 | 89 | 1,571 | 6.3 | 6.0 | 145 | 33 | 5,493 | 40,845 | 107 | 85 |
| 800 | 12 | 1.8 | – | 6,373 | 14 | 10 | 535 | 80 | – | 159,337 | 246 | 159 |

The M1 and M2 results for $m = 800$ do not include the time of the phase I. The other problems did not require a phase I

**Table 4** Time per iteration and total time for randomly generated band SDPs with $n = 200$ and $m = 100$

| $w$ | Time per iteration (s) | | | | | | Total time (s) | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | M1 | M2 | DSDP | SDPA-C | SDPT3 | SEDUMI | M1 | M2 | DSDP | SDPA-C | SDPT3 | SEDUMI |
| 0 | 0.03 | 0.02 | 0.39 | 3.3 | 0.07 | 0.02 | 1.4 | 0.65 | 20 | 95 | 1.2 | 0.20 |
| 1 | 0.10 | 0.05 | 2.1 | 8.4 | 0.24 | 0.31 | 4.4 | 2.0 | 109 | 235 | 4.0 | 4.7 |
| 2 | 0.14 | 0.06 | 1.2 | 10 | 0.28 | 0.33 | 4.9 | 2.2 | 42 | 259 | 4.5 | 4.6 |
| 4 | 0.19 | 0.09 | 1.2 | 17 | 0.41 | 0.36 | 7.3 | 3.4 | 41 | 393 | 6.5 | 4.7 |
| 8 | 0.34 | 0.16 | 1.2 | 31 | 0.66 | 0.40 | 13 | 6.0 | 39 | 782 | 9.9 | 5.6 |
| 16 | 0.71 | 0.33 | 2.2 | 65 | 1.2 | 0.50 | 26 | 12 | 73 | 1,355 | 16 | 7.0 |
| 32 | 1.7 | 0.84 | 2.2 | 127 | 0.87 | 0.68 | 69 | 34 | 74 | 2,660 | 11 | 8.2 |

M1 and M2 required a phase I for $w = 0$ and $w = 1$ (time not included)

In the next part of the experiment we fix $n$ and $w$, and vary the number of constraints $m$ (Table 3). Here M2 is the fastest method, and for these problems it scales much better than M1. We remark that DSDP did not return a solution for $m = 800$ (possibly due to insufficient memory). Again, the poor performance of SDPA-C on these chordal problems is likely a consequence of the technique used to compute the Schur complement matrix rather than a shortcoming of the completion technique.

Finally we consider a set of SDPs with different bandwidths and with $n$ and $m$ fixed. From the results in Table 4 it is clear that, for most of the solvers, there is a gap in time per iteration from $w = 0$ (i.e., a linear programming problem cast as an SDP) to $w = 1$ (i.e., a tridiagonal sparsity pattern). The gap is less evident for the chordal methods. Interestingly, SDPT3 becomes slightly faster when the bandwidth is increased from 16 to 32.

## 6.2 Matrix norm minimization

In the next experiment we consider a family of matrix norm minimization problems of the form

**Table 5** Time per iteration and total time for randomly generated dense matrix norm problems of size $p \times q$ with $r$ variables, for $q = 10, r = 100$

| $p+q$ | Time per iteration (s) | | | | | | Total time (s) | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | M1 | M2 | DSDP | SDPA-C | SDPT3 | SEDUMI | M1 | M2 | DSDP | SDPA-C | SDPT3 | SEDUMI |
| 100 | 0.23 | 0.12 | 0.09 | 0.91 | 0.29 | 0.09 | 6.2 | 3.1 | 3.1 | 21 | 3.5 | 1.2 |
| 200 | 0.48 | 0.24 | 0.39 | 2.1 | 0.78 | 0.34 | 13 | 6.3 | 8.2 | 55 | 9.3 | 4.4 |
| 400 | 1.0 | 0.51 | 1.8 | 5.5 | 2.5 | 1.8 | 29 | 14 | 38 | 132 | 32 | 25 |
| 800 | 2.1 | 1.1 | 10 | 15 | 9.1 | 15 | 60 | 32 | 219 | 437 | 136 | 223 |
| 1,600 | 4.3 | 2.5 | 77 | 43 | 35 | 162 | 146 | 84 | 1,772 | 1,372 | 491 | 2,424 |

$$\text{minimize} \quad \|F(x) + G\|_2 \tag{32}$$

where $F(x) = x_1F_1 + \cdots + x_rF_r$, and with randomly generated problem data $G, F_i \in \mathbf{R}^{p\times q}$. The matrix $G$ is dense while the sparsity of each $F_i$ is determined by a sparsity parameter $d \in (0, 1]$. The number of nonzero elements $F_i$ is given by $\max(1, \text{round}(dpq))$, and the locations of the nonzero elements are selected at random for each $F_i$. All nonzero elements are randomly generated from a normal distribution.

The matrix norm minimization problem (32) can be formulated as an SDP:

$$\begin{array}{ll} \text{minimize} & t \\ \text{subject to} & \begin{bmatrix} tI & F(x)+G \\ (F(x)+G)^T & tI \end{bmatrix} \succeq 0. \end{array} \tag{33}$$

The variables are $x \in \mathbf{R}^r$ and $t \in \mathbf{R}$. Since $G$ is dense, the aggregate sparsity pattern is independent of the sparsity parameter $d$. Moreover, this aggregate sparsity pattern is nonchordal for $q > 1$, but a chordal embedding is easily obtained by filling the smaller of the two diagonal blocks in the aggregate sparsity pattern associated with (33). In the special case where $q = 1$, the SDP (33) is equivalent to a second-order cone program.

Of the $r + 1$ data matrices associated with the SDP (33), one matrix always has full rank (the identity matrix) whereas the rank of each of the remaining $r$ data matrices is at most $\min(2p, 2q)$. This means that low-rank structure may be exploited when $p \gg q$ or $p \ll q$.

In the first part of the experiment we look at the time per iteration for increasing values of $p$ and with $q$ and $r$ fixed. The data matrices $F_i$ are dense. The results in Table 5 verify that the time per iteration is roughly linear in $n = p+q$ for M1 and M2. This is shown in Fig. 4. Indeed, for large $p + q$, M1 and M2 are significantly faster in terms of time per iteration than the other interior-point solvers.

In the second part of the experiment we use matrix norm SDPs with $F_i$ dense, a varying number of constraints $m = r + 1$, and $p$ and $q$ fixed. Table 6 summarizes the results. Notice that M2 performs quite well, and it scales much better than M1. The main reason is that M1 will use Technique 1 to compute the columns of $H$ since the data matrices $F_i$ are dense (i.e., the number of nonzero columns in $A_i$ is $n = p + q$,

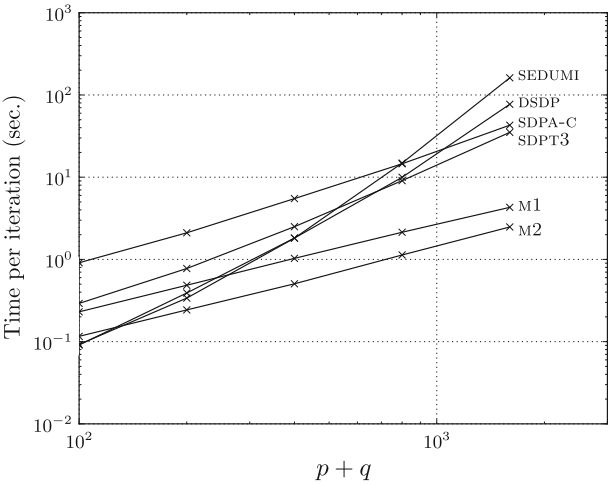


**Fig. 4** Time per iteration as a function of $p+q$. The iteration complexity is roughly linear for M1 and M2

**Table 6** Time per iteration and total time for randomly generated dense matrix norm problems of size $p \times q$ with $r$ variables, with $p = 400, q = 10$

| $r$ | Time per iteration (s) | | | | | | Total time (s) | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | M1 | M2 | DSDP | SDPA-C | SDPT3 | SEDUMI | M1 | M2 | DSDP | SDPA-C | SDPT3 | SEDUMI |
| 50 | 0.51 | 0.23 | 1.0 | 1.8 | 1.4 | 1.8 | 15 | 6.9 | 20 | 47 | 19 | 25 |
| 100 | 1.1 | 0.52 | 1.9 | 5.6 | 2.6 | 2.0 | 33 | 16 | 40 | 140 | 36 | 30 |
| 200 | 2.7 | 1.0 | 4.0 | 19 | 5.2 | 2.3 | 70 | 27 | 80 | 471 | 72 | 30 |
| 400 | 7.8 | 2.2 | 8.5 | 69 | 11 | 3.3 | 204 | 57 | 169 | 1,523 | 141 | 43 |
| 800 | 26 | 4.9 | 15 | 262 | 24 | 6.6 | 682 | 128 | 434 | 5,756 | 291 | 80 |

and as a consequence, Technique 2 is expensive). Notice also that SDPA-C is slow for large $m$. This can be attributed to the fact that SDPA-C uses an algorithm similar to Technique 2 to compute the Schur complement matrix $H$, and this is expensive whenever the data matrices have a large number of nonzero columns.

The effect of varying the density of $F_i$ can be observed in Table 7 which shows the time per iteration for matrix norm problems with varying density and fixed dimensions. For $d$ small, the data matrices $A_i$ generally have only a few nonzero columns, and this can be exploited by M1 and SDPA-C. For the two sparsest problems ($d < 0.01$), method M1 used technique T2 for all but one column in the Schur complement matrix; for the problems with $d \geq 0.01$, T1 was used for all columns. The times for M2 on the other hand are more or less independent of $d$ since M2 handles the data matrices $A_i$ as dense matrices in $\mathbf{S}^n_V$.

In the last part of this experiment we use matrix norm problems with different values of $q$, with $r$ and $p+q$ fixed, and dense problem data $F_i$. The results are shown in Table 8. Both M1 and M2 perform fairly well, but the time per iteration does not scale well with $q$. Observe that in the special case where $q = 1$, the chordal techniques

**Table 7** Time per iteration and total time for randomly generated sparse matrix norm problems of size $p \times q$ with $r$ variables and density $d$, for $p = 400$, $q = 10$, and $r = 200$

| $d$ | Time per iteration (s) | | | | | | Total time (s) | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | M1 | M2 | DSDP | SDPA-C | SDPT3 | SEDUMI | M1 | M2 | DSDP | SDPA-C | SDPT3 | SEDUMI |
| 0.001 | 0.31 | 0.96 | 0.09 | 0.22 | 0.22 | 1.7 | 9.2 | 29 | 2.0 | 4.9 | 2.9 | 24 |
| 0.005 | 0.69 | 0.97 | 0.28 | 0.42 | 0.33 | 1.7 | 19 | 26 | 6.5 | 10 | 4.6 | 26 |
| 0.01 | 1.2 | 0.97 | 0.36 | 0.53 | 0.60 | 1.8 | 34 | 28 | 8.2 | 13 | 8.4 | 26 |
| 0.02 | 1.2 | 0.96 | 0.96 | 0.71 | 1.6 | 1.8 | 33 | 27 | 21 | 16 | 21 | 24 |
| 0.05 | 1.2 | 0.95 | 1.6 | 1.2 | 2.3 | 1.8 | 40 | 31 | 36 | 30 | 30 | 23 |
| 0.1 | 1.3 | 0.96 | 1.5 | 2.1 | 2.5 | 1.8 | 37 | 28 | 34 | 49 | 33 | 26 |
| 0.25 | 1.5 | 0.98 | 2.5 | 4.7 | 3.2 | 1.9 | 39 | 25 | 54 | 117 | 48 | 27 |
| 0.5 | 1.9 | 1.0 | 3.0 | 9.6 | 4.0 | 2.0 | 50 | 26 | 67 | 232 | 56 | 28 |

**Table 8** Time per iteration and total time for randomly generated dense matrix norm problems of size $p \times q$ with $r$ variables, for $p + q = 1000$, $r = 10$

| $q$ | Time per iteration (s) | | | | | | Total time (s) | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | M1 | M2 | DSDP | SDPA-C | SDPT3 | SEDUMI | M1 | M2 | DSDP | SDPA-C | SDPT3 | SEDUMI |
| 1 | 0.10 | 0.05 | 1.6 | 0.19 | 1.5 | 28 | 3.2 | 1.5 | 18 | 3.4 | 15 | 142 |
| 2 | 0.12 | 0.05 | 1.7 | 0.32 | 1.7 | 28 | 3.6 | 1.7 | 31 | 5.7 | 22 | 305 |
| 5 | 0.25 | 0.12 | 1.9 | 0.72 | 1.7 | 27 | 7.4 | 3.5 | 36 | 18 | 22 | 356 |
| 10 | 0.51 | 0.23 | 2.3 | 1.4 | 2.8 | 26 | 15 | 6.8 | 44 | 32 | 39 | 397 |
| 20 | 1.3 | 0.58 | 3.8 | 3.0 | 4.0 | 27 | 50 | 23 | 69 | 72 | 53 | 383 |
| 50 | 7.8 | 3.5 | 5.8 | 9.8 | 7.3 | 28 | 234 | 105 | 110 | 264 | 102 | 478 |

appear superior. However it should be noted that both SDPT3 and SEDUMI can handle second-order cone constraints directly with greater efficiency when the second-order cone constraints are explicitly specified as such.

### 6.3 Overlapping cliques

In this experiment we consider a family of SDPs with chordal sparsity, obtained by modifying a block diagonal sparsity pattern so that neighboring blocks overlap. Let $l$ be the number of cliques, all of order $N$, and denote with $u$ the overlap between neighboring cliques. Note that $u$ must satisfy $0 \leq u \leq N - 1$ where $u = 0$ corresponds to a block diagonal sparsity pattern whereas $u = N - 1$ corresponds to a banded sparsity pattern with bandwidth $2u + 1$. The order of $V$ is $n = l(N - u) + u$, and the $l$ cliques are given by

$$W_i = \{(i-1)(N-u)+1, \ldots, i(N-u)+u\}, \quad i = 1, \ldots, l.$$

For the experiment we have chosen $l = 50$ cliques of order $N = 16$ and $m = 100$ constraints. The aggregate sparsity pattern as a function of the clique overlap $u$ is

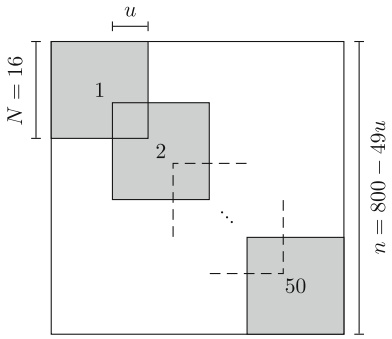


**Fig. 5** Sparsity pattern with 50 overlapping cliques of order 16. The parameter $u$ is the overlap between adjacent cliques

**Table 9** Time per iteration and total time for random SDPs with $l = 50$ cliques of order $N = 16$ and $m = 100$ constraints

| $u$ | Time per iteration (s) | | | | | | Total time (s) | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | M1 | M2 | DSDP | SDPA-C | SDPT3 | SEDUMI | M1 | M2 | DSDP | SDPA-C | SDPT3 | SEDUMI |
| 0 | 0.23 | 0.31 | 0.16 | 47 | 0.32 | 0.12 | 10 | 14 | 2.9 | 1,027 | 4.2 | 1.7 |
| 1 | 0.28 | 0.35 | 79 | 43 | 3.8 | 12 | 11 | 14 | 3,415 | 1,035 | 61 | 181 |
| 2 | 0.28 | 0.34 | 100 | 38 | 3.5 | 9.5 | 12 | 14 | 4,515 | 920 | 56 | 143 |
| 4 | 0.26 | 0.32 | 124 | 31 | 2.9 | 6.5 | 10 | 13 | 5,817 | 776 | 43 | 97 |
| 8 | 0.26 | 0.27 | 55 | 21 | 1.5 | 2.2 | 9.7 | 10 | 2,621 | 530 | 24 | 37 |
| 15 | 0.12 | 0.08 | 0.08 | 0.60 | 0.12 | 0.05 | 3.5 | 2.4 | 1.7 | 11 | 1.5 | 0.60 |

Neighboring cliques overlap with $u$ nodes

illustrated in Fig. 5. Each of the data matrices $A_i$ has roughly 10% nonzeros (relative to the aggregate sparsity pattern). Table 9 shows the results for different values of $u$. Not surprisingly, the nonchordal solvers all do quite well in the block diagonal case ($u = 0$), but there is a significant jump in complexity for these solvers when the cliques overlap just slightly and thereby destroy the block structure. On the other hand, M1 and M2 appear to be much less sensitive to increasing overlaps. We remark that M1 used only T1 for these problems. When $u = 15$ the sparsity pattern is banded and has order $n = 65$. This is a relatively small problem that can be solved quite easily with symmetric primal–dual methods.

We should point out that the conversion method by Fukuda et al. [18], which also exploits chordal structure, can be a viable alternative for this type of sparsity pattern when the clique overlaps are small and when the number of cliques is not too large. The conversion method can be implemented as a preprocessing step and used in conjunction with existing primal–dual interior-point codes.

### 6.4 Sparse SDPs from SDPLIB

Our next experiment is based on a set of problem instances from SDPLIB [6]. We include only the largest and most sparse problem instances (specifically, sparse

problems with $n \geq 500$) since these are the most interesting problems with respect to chordal matrix techniques. Before we present and discuss our results, we briefly give some definitions pertaining to problem dimension and data sparsity.

Suppose $V$ is a sparsity pattern with $k$ diagonal blocks of order $n_1, \ldots, n_k$, where block $i$ has sparsity pattern $V_i$. In other words, $\mathbf{S}_V^n = \mathbf{S}_{V_1}^{n_1} \times \cdots \times \mathbf{S}_{V_k}^{n_k}$ and $n = \sum_{i=1}^k n_i$. We denote with $n_{\max} = \max_i n_i$ the order of the largest block. We define the *density* of $V$ as

$$\rho_V = \frac{2|V| - n}{\sum_{i=1}^k n_i^2}.$$

(Recall that $|V|$ is the number of lower triangular nonzeros in the sparsity pattern.) Note that $\rho_V = 1$ corresponds to dense blocks in which case the chordal techniques become trivial. Similarly we define the *average density* of the problem data relative to $V$ as

$$\bar{\rho}_{V,\mathrm{rel}} = \frac{1}{m} \sum_{i=1}^m \frac{\mathrm{nnz}(A_i)}{2|V| - n},$$

where $\mathrm{nnz}(A_i)$ is the number of nonzeros in $A_i$. This is a measure of the sparsity of the coefficients $A_i$ in the subspace $\mathbf{S}_V^n$. A value $\bar{\rho}_{V,\mathrm{rel}} = 1$ implies that the coefficient matrices are all dense relative to $\mathbf{S}_V^n$. Note that M2 is generally inefficient when $\bar{\rho}_{V,\mathrm{rel}}$ is small since M2 treats $A_i$ as a dense element in $\mathbf{S}_V^n$.

For a chordal sparsity pattern with $l$ cliques we define the following two measures:

$$\mathcal{W} = \sum_{i=1}^l |W_i| \quad \text{and} \quad \mathcal{U} = \sum_{i=1}^l |U_i|.$$

$\mathcal{W}$ is the sum of clique cardinalities, and $\mathcal{U}$ is the sum of separator cardinalities. The sum of residual cardinalities is given by $\sum_{i=1}^l |V_i| = \mathcal{W} - \mathcal{U} = n$, and hence does not carry any information about the sparsity pattern. The measure $\mathcal{U}$ can be thought of as the total overlap between cliques, where $\mathcal{U} = 0$ corresponds to nonoverlapping cliques, i.e., $V$ is block diagonal with dense blocks. Finally we denote with $w_{\max}$ the order of the maximum clique in $V$.

The set of SDPLIB problems with $n \geq 500$ is listed in Table 10 with selected problem statistics. With the exception of one problem (`truss8`), all problem instances have a nonchordal aggregate sparsity pattern. The problem `truss8` clearly has a block diagonal aggregate sparsity pattern with dense blocks (since $\rho_V = 1$). This implies that the chordal techniques are trivial and have no advantage over existing solvers that handle block diagonal structure. Notice also that $\bar{\rho}_{V,\mathrm{rel}}$ is small for all problem instances, and as a consequence, M2 can be expected to be quite slow since it does not fully exploit the sparsity of the individual data matrices. Finally we remark that all the problem instances in Table 10 have data matrices with very low rank.

Table 11 shows some statistics for two embeddings: a standard AMD embedding and an AMD embedding obtained with CHOLMOD [12] which includes a

**Table 10** Problem statistics for SDPLIB problems

| Instance | Dimensions | | | | Sparsity (%) | |
|---|---|---|---|---|---|---|
| | $m$ | $n$ | $k$ | $n_{\max}$ | $\rho_V$ | $\bar{\rho}_{V,\text{rel}}$ |
| maxG11 | 800 | 800 | 1 | 800 | 0.62 | 0.025 |
| maxG32 | 2,000 | 2,000 | 1 | 2,000 | 0.25 | 0.010 |
| maxG51 | 1,000 | 1,000 | 1 | 1,000 | 1.28 | 0.008 |
| maxG55 | 5,000 | 5,000 | 1 | 5,000 | 0.14 | 0.003 |
| maxG60 | 7,000 | 7,000 | 1 | 7,000 | 0.08 | 0.002 |
| mcp500-1 | 500 | 500 | 1 | 500 | 0.70 | 0.057 |
| mcp500-2 | 500 | 500 | 1 | 500 | 1.18 | 0.034 |
| mcp500-3 | 500 | 500 | 1 | 500 | 2.08 | 0.019 |
| mcp500-4 | 500 | 500 | 1 | 500 | 4.30 | 0.009 |
| qpG11 | 800 | 1,600 | 1 | 1,600 | 0.19 | 0.042 |
| qpG51 | 1,000 | 2,000 | 1 | 2,000 | 0.35 | 0.014 |
| thetaG11 | 2,401 | 801 | 1 | 801 | 0.87 | 0.113 |
| thetaG51 | 6,910 | 1,001 | 1 | 1,001 | 1.48 | 0.053 |
| truss8 | 496 | 628 | 34 | 19 | 100.00 | 0.270 |

**Table 11** Statistics for two chordal embeddings of selected SDPLIB sparsity patterns

| Instance | AMD embedding | | | | | CHOLMOD-AMD embedding | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | $l$ | $w_{\max}$ | $\mathcal{W}$ | $\mathcal{U}$ | $\rho_V$ (%) | $l$ | $w_{\max}$ | $\mathcal{W}$ | $\mathcal{U}$ | $\rho_V$ (%) |
| maxG11 | 598 | 24 | 4,552 | 3,752 | 2.48 | 134 | 32 | 1,864 | 1,064 | 4.92 |
| maxG32 | 1,498 | 76 | 12,984 | 10,984 | 1.81 | 253 | 79 | 5,348 | 3,348 | 3.12 |
| maxG51 | 674 | 326 | 14,286 | 13,286 | 13.41 | 129 | 337 | 4,563 | 3,563 | 20.81 |
| maxG55 | 3,271 | 1,723 | 77,908 | 72,908 | 12.55 | 728 | 1,776 | 28,421 | 23,421 | 15.30 |
| maxG60 | 5,004 | 1,990 | 100,163 | 93,163 | 8.51 | 1,131 | 2,048 | 35,817 | 28,817 | 10.27 |
| mcp500-1 | 452 | 39 | 1,911 | 1,411 | 2.07 | 127 | 51 | 860 | 360 | 5.55 |
| mcp500-2 | 363 | 138 | 4,222 | 3,722 | 10.74 | 99 | 146 | 1,545 | 1,045 | 18.71 |
| mcp500-3 | 259 | 242 | 6,072 | 5,572 | 27.99 | 58 | 251 | 2,196 | 1,696 | 42.08 |
| mcp500-4 | 161 | 340 | 8,420 | 7,920 | 52.64 | 32 | 352 | 2,608 | 2,108 | 68.60 |
| qpG11 | 1,398 | 24 | 5,352 | 3,752 | 0.65 | 934 | 32 | 2,664 | 1,064 | 1.26 |
| qpG51 | 1,674 | 326 | 15,286 | 13,286 | 3.38 | 1,129 | 337 | 5,563 | 3,563 | 5.23 |
| thetaG11 | 598 | 25 | 5,150 | 4,349 | 2.72 | 134 | 33 | 1,998 | 1,197 | 5.16 |
| thetaG51 | 676 | 324 | 14,883 | 13,882 | 13.41 | 127 | 335 | 4,707 | 3,706 | 20.97 |
| truss8 | 34 | 19 | 628 | 0 | 100.00 | 34 | 19 | 628 | 0 | 100.00 |

post-processing of the clique tree. CHOLMOD's embedding clearly has fewer but larger cliques, and therefore the density $\rho_V$ is somewhat larger. We will refer to Method 1 based on CHOLMOD's AMD embedding as M1C.

**Table 12** Average time per iteration and total time for selected SDPLIB problems

| Instance | Time per iteration (s) | | | | | | Total time (s) | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | M1 | M1C | DSDP | SDPA-C | SDPT3 | SEDUMI | M1 | M1C | DSDP | SDPA-C | SDPT3 | SEDUMI |
| maxG11 | 0.74 | 0.47 | 0.24 | 0.48 | 0.98 | 13 | 23 | 16 | 5.0 | 11 | 15 | 169 |
| maxG32 | 5.1 | 3.3 | 3.2 | 5.1 | 9.5 | 339 | 154 | 118 | 70 | 133 | 142 | 4743 |
| maxG51 | 8.3 | 5.4 | 0.64 | 2.8 | 2.0 | 31 | 282 | 196 | 19 | 76 | 34 | 498 |
| maxG55 | 1,169 | 650 | 73 | 852 | 149 | m | 45,598 | 28,590 | 2,618 | 22,146 | 2,539 | m |
| maxG60 | 2,301 | 1,074 | 170 | 1,883 | 428 | m | 92,029 | 49,398 | 6,131 | 45,194 | 6,848 | m |
| mcp500-1 | 0.33 | 0.18 | 0.07 | 0.11 | 0.29 | 3.1 | 9.5 | 6.5 | 1.9 | 2.3 | 4.4 | 50 |
| mcp500-2 | 1.3 | 0.77 | 0.10 | 0.33 | 0.37 | 3.3 | 32 | 19 | 2.2 | 6.9 | 5.9 | 50 |
| mcp500-3 | 3.6 | 2.0 | 0.14 | 0.86 | 0.42 | 3.3 | 109 | 64 | 2.7 | 18 | 5.9 | 50 |
| mcp500-4 | 13 | 4.4 | 0.20 | 1.7 | 0.47 | 3.4 | 349 | 124 | 4.2 | 37 | 6.1 | 47 |
| qpG11 | 1.1 | 0.76 | 0.38 | 0.77 | 0.97 | 158 | 33 | 30 | 12 | 18 | 15 | 2214 |
| qpG51 | 12 | 8.4 | 0.97 | 4.6 | 1.9 | 308 | 682 | 488 | 95 | 252 | 32 | 6,775 |
| thetaG11 | 4.2 | 3.2 | 3.2 | 3.4 | 2.7 | 21 | 248 | 178 | 200 | 67 | 49 | 317 |
| thetaG51 | 59 | 52 | 94 | 85 | 33 | 301 | 2,901 | 2,676 | 5,662 | 5,726 | 1,223 | 5,422 |
| truss8 | 0.60 | 0.60 | 0.14 | 2.6 | 0.17 | 0.13 | 26 | 26 | 3.7 | 87 | 2.8 | 3.1 |

Failure due to insufficient memory is marked with an 'm'. M1 and M1C required a phase I for the problems thetaG11, thetaG51, and truss8 (time not included)

If we compare the density of the aggregate sparsity patterns (Table 10) with the density of the chordal embeddings (Table 11) we see that for some of the problems, the chordal embeddings are much more dense than the aggregate sparsity pattern. This means that the chordal embeddings are not very efficient. Indeed this seems to be the case for many of the problems and in particular maxG55, maxG60, mcp500-3, and mcp500-4. Notice also that the chordal embeddings of maxG55 and maxG60 have quite large maximum clique sizes.

From the results in Table 12 we see that the performance of the chordal techniques varies quite a bit when compared to the nonchordal solvers. As expected, the problems maxG55, maxG60, mcp500-3, and mcp500-4 are not favorable to the chordal techniques. Problems with more efficient chordal embeddings and smaller maximum clique sizes, such as maxG11 and maxG32, are solved more efficiently. If we compare M1 and M1C, it is readily seen that the chordal embedding obtained with CHOLMOD consistently results in faster iteration times. Other embedding techniques can be used and may further improve the speed. Notice that DSDP does quite well on most problems, and this may partially be due to DSDP's use of low rank techniques. Furthermore, DSDP solves the Schur complement equations using preconditioned conjugate gradient which is often more efficient, especially when $m$ is large and while the condition number of the Schur complement is not too large. The chordal techniques do not exclude the use of low rank techniques or iterative solution of the Newton system, so implementing these techniques may also improve M1/M1C.

Finally we remark that for the problem thetaG51 all the solvers stopped prematurely because of numerical problems, and for the problems maxG55 and maxG60, SDPA-C spent more than 50% of the total CPU time outside the main loop.

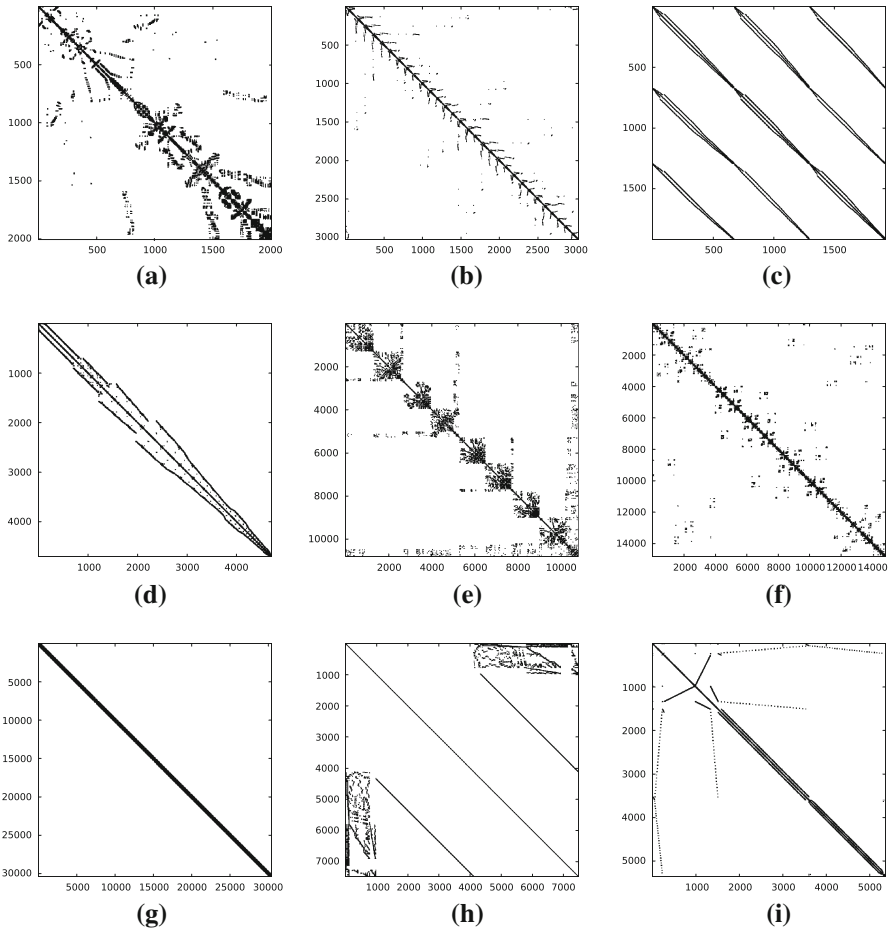


**Fig. 6** Aggregate sparsity patterns for nonchordal test problems: **a** `rs35`, **b** `rs200`, **c** `rs228`, **d** `rs365`, **e** `rs828`, **f** `rs1184`, **g** `rs1288`, **h** `rs1555`, **i** `rs1907`

### 6.5 Nonchordal SDPs

In our last experiment we consider SDPs with random data and nonchordal aggregate sparsity patterns. Each problem is based on a sparsity pattern from the University of Florida Sparse Matrix Collection [13]. We will use as problem identifier the name `rsX` where `X` is the ID number associated with the corresponding problem from [13]. Figure 6 shows a selection of nine nonchordal sparsity patterns used in this experiment.

For each of the sparsity patterns we generated a problem instance with average density $\bar{\rho}_{V,\mathrm{rel}} = 10^{-3}$ and $m = 200$ constraints. Table 13 lists problem statistics and Table 14 shows some statistics for two different chordal embeddings.

All nine problems have a single block, and therefore for the largest of the problems, handling the primal variable as a dense matrix is prohibitively expensive in terms of both computations and memory. However, for all nine problems the chordal embeddings have maximum clique sizes that are much smaller than $n$. Notice that

**Table 13** Problem statistics for nonchordal problems

| Instance | Dimensions | | | | Sparsity (%) | |
|---|---|---|---|---|---|---|
| | $m$ | $n$ | $k$ | $n_{\max}$ | $\rho_V$ | $\bar{\rho}_{V,\mathrm{rel}}$ |
| rs35 | 200 | 2,003 | 1 | 2,003 | 2.09 | 0.05 |
| rs200 | 200 | 3,025 | 1 | 3,025 | 0.23 | 0.05 |
| rs228 | 200 | 1,919 | 1 | 1,919 | 0.88 | 0.05 |
| rs365 | 200 | 4,704 | 1 | 4,704 | 0.47 | 0.05 |
| rs828 | 200 | 10,800 | 1 | 10,800 | 0.69 | 0.05 |
| rs1184 | 200 | 14,822 | 1 | 14,822 | 0.33 | 0.05 |
| rs1288 | 200 | 30,401 | 1 | 30,401 | 0.05 | 0.05 |
| rs1555 | 200 | 7,479 | 1 | 7,479 | 0.12 | 0.05 |
| rs1907 | 200 | 5,357 | 1 | 5,357 | 0.72 | 0.05 |

**Table 14** Statistics for AMD and CHOLMOD's AMD embeddings of nonchordal problems

| Instance | AMD embedding | | | | | CHOLMOD-AMD embedding | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | $l$ | $w_{\max}$ | $\mathcal{W}$ | $\mathcal{U}$ | $\rho_V$ (%) | $l$ | $w_{\max}$ | $\mathcal{W}$ | $\mathcal{U}$ | $\rho_V$ (%) |
| rs35 | 589 | 343 | 27,881 | 25,878 | 13.21 | 141 | 394 | 11,819 | 9,816 | 14.66 |
| rs200 | 1,632 | 95 | 20,063 | 17,038 | 1.51 | 314 | 95 | 8,711 | 5,686 | 2.12 |
| rs228 | 790 | 88 | 17,244 | 15,325 | 3.60 | 207 | 88 | 7,333 | 5,414 | 4.35 |
| rs365 | 1,230 | 296 | 36,136 | 31,432 | 2.54 | 396 | 296 | 19,417 | 14,713 | 2.87 |
| rs828 | 841 | 480 | 74,256 | 63,456 | 2.19 | 605 | 480 | 56,364 | 45,564 | 2.25 |
| rs1184 | 2,236 | 500 | 172,046 | 157,224 | 2.28 | 830 | 500 | 91,966 | 77,144 | 2.36 |
| rs1288 | 10,394 | 412 | 222,706 | 192,305 | 0.32 | 2,295 | 412 | 108,865 | 78,464 | 0.38 |
| rs1555 | 6,891 | 184 | 49,447 | 41,968 | 0.28 | 2,371 | 193 | 23,614 | 16,135 | 0.55 |
| rs1907 | 577 | 261 | 30,537 | 25,180 | 2.97 | 400 | 261 | 24,974 | 19,617 | 3.10 |

CHOLMOD's AMD embedding generally has significantly fewer cliques than the AMD embedding, and its density is only slightly higher. The chordal embeddings have between 3 and 10 times as many nonzeros as the corresponding aggregate sparsity patterns.

The results are listed in Table 15. For three of the four smallest problem, DSDP is the fastest in terms of average time per iteration. The results show that the chordal techniques can be quite fast for large problems even when the chordal embedding has many more nonzeros than the aggregate sparsity pattern. If we compare the results obtained with M1 and M1C, we see that CHOLMOD's chordal embedding is advantageous in terms of iteration time, but the difference in iteration time is typically smaller than for many of the SDPLIB problems. Both M1 and M1C used only T2 to compute the Schur complement matrix in all cases.

The general purpose primal–dual solvers SDPT3 and SEDUMI ran of of memory while solving the largest of the problems, and while DSDP successfully solved the largest instance rs1288, it ultimately ran out of memory in an attempt to compute the dense primal variable.

**Table 15** Average time per iteration and total time for nonchordal SDPs with random data

| Instance | Time per iteration (s) | | | | | | Total time (s) | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | M1 | M1C | DSDP | SDPA-C | SDPT3 | SEDUMI | M1 | M1C | DSDP | SDPA-C | SDPT3 | SEDUMI |
| rs35 | 37 | 30 | 4.0 | 41 | 13 | 325 | 1,632 | 1,377 | 205 | 1,245 | 234 | 5,530 |
| rs200 | 3.2 | 2.4 | 2.9 | 5.9 | 33 | 1,139 | 171 | 142 | 117 | 196 | 727 | 19,367 |
| rs228 | 3.9 | 3.0 | 1.4 | 5.7 | 11 | 282 | 187 | 139 | 71 | 181 | 214 | 5,914 |
| rs365 | 30 | 28 | 15 | 59 | 100 | m | 1473 | 1368 | 850 | 1,940 | 1,993 | m |
| rs828 | 406 | 392 | 482 | 1,813 | m | m | 21,107 | 20,401 | 27,000 | 63,448 | m | m |
| rs1184 | 729 | 677 | 791 | 2,925 | m | m | 43,010 | 37,897 | 47,460 | 105,313 | m | m |
| rs1288 | 386 | 362 | m | 2,070 | m | m | 23,548 | 20,621 | m | 74,528 | m | m |
| rs1555 | 14 | 11 | 22 | 23 | m | m | 641 | 484 | 1062 | 648 | m | m |
| rs1907 | 49 | 48 | 38 | 176 | 152 | m | 2,531 | 2,192 | 2,004 | 5,445 | 2888 | m |

Failure due to insufficient memory is marked with an 'm'

## 7 Summary and conclusions

We have discussed interior-point methods for linear cone programs involving two types of sparse matrix cones: cones of positive semidefinite matrices with a given chordal sparsity pattern, and their associated dual cones, which consist of the chordal sparse matrices that have a positive semidefinite completion. These cones include as special cases the familiar cones in linear, second-order cone, and semidefinite programming, i.e., the nonnegative orthant (equivalent to a matrix cone with diagonal sparsity pattern), the second-order cone (equivalent to a matrix cone with arrow pattern), and the positive semidefinite cone. They also include a variety of useful nonsymmetric cones, for example, cones of sparse matrices with band or block-arrow patterns, as well as chordal embeddings of general (non-chordal) sparse matrix cones.

Sparse matrix cone programs are usually solved as semidefinite programs, i.e., by embedding the cones into dense positive semidefinite cones. Sparsity in the coefficient matrices is exploited, to the extent possible, when solving the Newton equations. The advantage of this approach is that symmetric primal–dual methods can be applied. An alternative approach is to solve the sparse matrix cone programs directly via a nonsymmetric (primal or dual) interior-point method. This makes it possible to take advantage of efficient algorithms for computing the values, and first and second derivatives of the logarithmic barriers for the chordal sparse matrix cones. It is not clear a priori which of the two approaches is better. A theoretical comparison is difficult because the techniques for exploiting sparsity in SDP solvers are quite involved and their efficiency depends greatly on the sparsity pattern. The main goal of this work was to answer the question via an experimental study, by developing a new solver for chordal sparse matrix cone programs, and comparing its performance with the publicly available interior-point solvers for semidefinite programming.

We have implemented a primal and a dual path-following algorithm that take advantage of chordal sparsity when solving the Newton equations. Two different methods were implemented for solving the Newton equations. The first method forms and solves

the Schur complement system via a Cholesky factorization. This method can exploit the sparsity of the individual data matrices more aggressively, but is also less stable than the second method. The second method avoids the explicit construction of the Schur complement matrix, by solving the augmented system via a QR decomposition. Although the augmented system approach is rarely practical in general semidefinite programming because of its high cost (and despite its better numerical stability), it can be viable for matrix cones of relatively low dimension. The two methods can be used in any interior-point method based on primal or dual scaling, including barrier and potential reduction methods, infeasible path-following methods, and Nesterov's nonsymmetric primal–dual methods [37].

Several of the ideas described in the paper have been proposed elsewhere. For example, the technique for evaluating the dual Hessian in Srijuntongsiri and Vavasis's algorithm [48] is essentially the same as method 1 in Sect. 4 of this paper. Other techniques, for example, method 2 in Sect. 4, appear to be new.

The results of the experiments indicate that if the sparsity patterns are chordal, the nonsymmetric chordal methods are often significantly faster and can solve larger instances than the SDP solvers. For problems with a band pattern, for example, the complexity of the chordal methods increases linearly with the matrix dimension, while the complexity of the other interior-point SDP solvers increases quadratically or faster. For general (non-chordal) sparsity patterns, the performance depends on the efficiency of the chordal embedding. If no efficient chordal embedding can be found, the chordal techniques are generally not advantageous, although in many cases their efficiency is still comparable with the primal–dual interior-point solvers.

A second conclusion is that the chordal techniques can serve as a complement but not a replacement of the various techniques used in existing SDP solvers. In our implementation of method 1, for example, we found it useful to exploit sparsity in the data matrices using a technique (T2) that does not rely on chordality, and is related to techniques common in sparse semidefinite programming. As another example, the good results for DSDP show the importance of exploiting low-rank structure in many applications.

We can mention a few possible directions for continuing this research. As already pointed out, the efficiency and robustness of the solver can be improved in many respects, for example, by a better selection of starting point, the use of a self-dual embedding for initialization, or by using a more sophisticated predictor steps. It would also be of interest to explore the use of chordal techniques for preconditioning iterative algorithms for the Newton equations. Several parts of the implementation are amenable to parallelization with, e.g., OpenMP. Other development plans include the creation of a stand-alone C library.



## References

1. Amestoy, P., Davis, T., Duff, I.: An approximate minimum degree ordering. SIAM J. Matrix Anal. Appl. **17**(4), 886–905 (1996)

2. Alizadeh, F., Goldfarb, D.: Second-order cone programming. Math. Program. Ser. B **95**, 3–51 (2003)
3. Alizadeh, F., Haeberly, J.-P.A., Overton, M.L.: Primal–dual interior-point methods for semidefinite programming: convergence rates, stability and numerical results. SIAM J. Optim. **8**(3), 746–768 (1998)
4. Barrett, W.W., Johnson, C.R., Lundquist, M.: Determinantal formulation for matrix completions associated with chordal graphs. Linear Algebra Appl. **121**, 265–289 (1989)
5. Borchers, B.: CSDP, a C library for semidefinite programming. Optim. Methods Softw. **11**(1), 613–623 (1999)
6. Borchers, B.: SDPLIB 1.2, a library of semidefinite programming test problems. Optim. Methods Soft. **11**(1), 683–690 (1999)
7. Blair, J.R.S., Peyton, B.: An introduction to chordal graphs and clique trees. In: George, A., Gilbert, J.R., Liu, J.W.H. Graph Theory and Sparse Matrix Computation, Springer, Berlin (1993)
8. Ben-Tal, A., Nemirovski, A.: Lectures on Modern Convex Optimization. Analysis, Algorithms, and Engineering Applications. Society for Pure and Applied Mathematics (2001)
9. Burer, S.: Semidefinite programming in the space of partial positive semidefinite matrices. SIAM J. Optim. **14**(1), 139–172 (2003)
10. Boyd, S., Vandenberghe, L.: Convex Optimization. Cambridge University Press (2004). http://www.stanford.edu/~boyd/cvxbook
11. Benson, S.J., Ye, Y.: DSDP5: Software for semidefinite programming. Technical Report ANL/MCS-P1289-0905, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, September 2005. Submitted to ACM Transactions on Mathematical Software
12. Chen, Y., Davis, T.A., Hager, W.W., Rajamanickam, S.: Algorithm 887: CHOLMOD, supernodal sparse Cholesky factorization and update/downdate. ACM Trans. Math. Softw. **35**(3), 1–14 (2008)
13. Davis, T.A.: The University of Florida Sparse Matrix Collection. Technical report, Department of Computer and Information Science and Engineering, University of Florida (2009)
14. Dahl, J., Vandenberghe, L.: CVXOPT: A Python Package for Convex Optimization. http://abel.ee.ucla.edu/cvxopt (2008)
15. Dahl, J., Vandenberghe, L.: CHOMPACK: Chordal Matrix Package. http://abel.ee.ucla.edu/chompack (2009)
16. Dahl, J., Vandenberghe, L., Roychowdhury, V.: Covariance selection for non-chordal graphs via chordal embedding. Optim. Methods Softw. **23**(4), 501–520 (2008)
17. El Ghaoui, L., Lebret, H.: Robust solutions to least-squares problems with uncertain data. SIAM J. Matrix Anal. Appl. **18**(4), 1035–1064 (1997)
18. Fukuda, M., Kojima, M., Murota, K., Nakata, K.: Exploiting sparsity in semidefinite programming via matrix completion. I. General framework. SIAM J. Optim. **11**, 647–674 (2000)
19. Fujisawa, K., Kojima, M., Nakata, K.: Exploiting sparsity in primal–dual interior-point methods for semidefinite programming. Math. Program. **79**(1–3), 235–253 (1997)
20. Fourer, R., Mehrotra, S.: Solving symmetric indefinite systems in an interior-point approach for linear programming. Math. Program. **62**, 15–39 (1993)
21. Grant, M., Boyd, S.: CVX: Matlab software for disciplined convex programming (web page and software). http://stanford.edu/~boyd/cvx (2007)
22. Grant, M., Boyd, S.: Graph implementations for nonsmooth convex programs. In: Blondel, V., Boyd, S., Kimura, H. (eds.) Recent Advances in Learning and Control (a tribute to M. Vidyasagar). Springer, Berlin (2008)
23. George, A.: Nested dissection of a regular finite element mesh. SIAM J. Numer. Anal. **10**(2), 345–363 (1973)
24. Goldfarb, D., Iyengar, G.: Robust convex quadratically constrained programs. Math. Program. Ser. B **97**, 495–515 (2003)
25. Grone, R., Johnson, C.R., Sá, E.M., Wolkowicz, H.: Positive definite completions of partial Hermitian matrices. Linear Algebra Appl. **58**, 109–124 (1984)
26. Hauser, R.A., Güler, O.: Self-scaled barrier functions on symmetric cones and their classification. Found. Comput. Math. **2**, 121–143 (2002)
27. Helmberg, C., Rendl, F., Vanderbei, R.J., Wolkowicz, H.: An interior-point method for semidefinite programming. SIAM J. Optim. **6**(2), 342–361 (1996)
28. Johnson, D., Pataki, G., Alizadeh, F.: Seventh DIMACS implementation challenge: Semidefinite and related problems (2000). http://dimacs.rutgers.edu/Challenges/Seventh
29. Kobayashi, K., Kim, S., Kojima, M.: Correlative sparsity in primal–dual interior-point methods for LP, SDP, and SOCP. Appl. Math. Optim. **58**(1), 69–88 (2008)

30. Kojima, M., Shindoh, S., Hara, S.: Interior-point methods for the monotone linear complementarity problem in symmetric matrices. SIAM J. Optim. **7**, 86–125 (1997)
31. Lauritzen, S.L.: Graphical Models. Oxford University Press, Oxford (1996)
32. Löfberg, J.: YALMIP : A Toolbox for Modeling and Optimization in MATLAB (2004)
33. Löfberg, J.: YALMIP : A toolbox for modeling and optimization in MATLAB. In: Proceedings of the CACSD Conference, Taipei, Taiwan (2004)
34. Monteiro, R.D.C.: Primal–dual path following algorithms for semidefinite programming. SIAM J. Optim. **7**, 663–678 (1995)
35. Monteiro, R.D.C.: Polynomial convergence of primal–dual algorithms for semidefinite programming based on Monteiro and Zhang family of directions. SIAM J. Optim. **8**(3), 797–812 (1998)
36. Nesterov, Yu.: Nonsymmetric potential-reduction methods for general cones. Technical Report 2006/34, CORE Discussion Paper, Université catholique de Louvain (2006)
37. Nesterov, Yu.: Towards nonsymmetric conic optimization. Technical Report 2006/28, CORE Discussion Paper, Université catholique de Louvain (2006)
38. Nakata, K., Fujitsawa, K., Fukuda, M., Kojima, M., Murota, K.: Exploiting sparsity in semidefinite programming via matrix completion. II. Implementation and numerical details. Math. Program. Ser. B **95**, 303–327 (2003)
39. Nesterov, Yu., Nemirovskii, A.: Interior-point polynomial methods in convex programming. Studies in Applied Mathematics, vol. 13. SIAM, Philadelphia (1994)
40. Nesterov, Yu.E., Todd, M.J.: Self-scaled barriers and interior-point methods for convex programming. Math. Oper. Res. **22**(1), 1–42 (1997)
41. Nesterov, Yu.E., Todd, M.J.: Primal–dual interior-point methods for self-scaled cones. SIAM J. Optim. **8**(2), 324–364 (1998)
42. Renegar, J.: A Mathematical View of Interior-Point Methods in Convex Optimization. Society for Industrial and Applied Mathematics (2001)
43. Rose, D.J.: Triangulated graphs and the elimination process. J. Math. Anal. Appl. **32**, 597–609 (1970)
44. Rose, D.J., Tarjan, R.E., Lueker, G.S.: Algorithmic aspects of vertex elimination on graphs. SIAM J. Comput. **5**(2), 266–283 (1976)
45. Sturm, J.F.: Using SEDUMI 1.02, a Matlab toolbox for optimization over symmetric cones. Optim. Methods Softw. **11–12**, 625–653 (1999)
46. Sturm, J.F.: Implementation of interior point methods for mixed semidefinite and second order cone optimization problems. Optim. Methods Softw. **17**(6), 1105–1154 (2002)
47. Sturm, J.F.: Avoiding numerical cancellation in the interior point method for solving semidefinite programs. Math. Program. Ser. B **95**, 219–247 (2003)
48. Srijuntongsiri, G., Vavasis, S.A.: A fully sparse implementation of a primal–dual interior-point potential reduction method for semidefinite programming (2004). Available at arXiv: arXiv:cs/0412009v1
49. Todd, M.J., Toh, K.C., Tütüncü, R.H.: On the Nesterov–Todd direction in semidefinite programming. SIAM J. Optim. **8**(3), 769–796 (1998)
50. Tütüncü, R.H., Toh, K.C., Todd, M.J.: Solving semidefinite-quadratic-linear programs using SDPT3. Math. Program. Ser. B **95**, 189–217 (2003)
51. Tarjan, R.E., Yannakakis, M.: Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. SIAM J. Comput. **13**(3), 566–579 (1984)
52. Vandenberghe, L., Boyd, S.: Semidefinite programming. SIAM Rev. **38**(1), 49–95 (1996)
53. Wermuth, N.: Linear recursive equations, covariance selection, and path analysis. J. Am Stat. Assoc. **75**(372), 963–972 (1980)
54. Waki, H., Kim, S., Kojima, M., Muramatsu, M.: Sums of squares and semidefinite program relaxations for polynomial optimization problems with structured sparsity. SIAM J. Optim. **17**(1), 218–241 (2006)
55. Wright, S.J.: Primal–Dual Interior-Point Methods. SIAM, Philadelphia (1997)
56. Yamashita, M., Fujisawa, K., Kojima, M.: Implementation and evaluation of SDPA 6.0 (Semidefinite Programming Algorithm 6.0). Optim. Methods Softw. **18**(4), 491–505 (2003)